UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Robin Emeršič

Mobilna aplikacija za prikaz prostorov in načrtovanje poti v stavbah

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2015

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Robin Emeršič

Mobilna aplikacija za prikaz prostorov in načrtovanje poti v stavbah

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Matjaž Kukar

Ljubljana, 2015

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Ob prihodu v novo, večjo stavbo (fakulteta, muzej, sejmišče) so obiskovalci pogosto zmedeni in izgubljeni. Pri tem jim lahko pomagajo mobilne aplikacije, ki mobilni telefon spremenijo v navigacijsko napravo. Vendar pa je navigacija znotraj stavb zaradi številnih dejavnikov bistveno trši oreh kot na prostem. Kandidat naj razvije mobilno aplikacijo, ki bo znotraj konkretne stavbe (nova stavba Fakultete za računalništvo in informatiko) omogočala iskanje osebja in prostorov, ter načrtovanje poti med dvema prostoroma. Aplikacija naj bo razvita za mobilni operacijski sistem Android in naj podpira čim več različnih naprav. Zasnovana naj bo splošno, tako da zamenjava stavbe ne predstavlja večjega problema. Ponuja naj vsaj omejeno pozicioniranje s pomočjo odčitavanja kod QR, ter kar se da enostavno posodabljanje.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Robin Emeršič sem avtor diplomskega dela z naslovom:

Mobilna aplikacija za prikaz prostorov in načrtovanje poti v stavbah

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matjaža Kukarja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Matjažu Kukarju za vse napotke in svetovanje pri izdelavi diplomskega dela. Prav tako se zahvaljujem družini in prijateljem, ki so mi ob študiju stali ob strani in me podpirali.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
CSV	Comma separated Values	Vrednosti ločene z vejico
GPS	Global Positioning System	Globalni pozicijski sistem
IDE	Integrated Development Environment	Integrirano razvojno okolje
JOSM	Java OpenStreetMap Editor	Urejevalnik zemljevidov OpenStreetMap
JSON	JavaScript Object Notation	Standardiziran zapis za prenos podatkov
POI	Point of interest	Točka zanimanja na zemljevidu
QR code	Quick Response Code	Koda za hitro prepoznavo
SDK	Software Development Kit	Komplet programskih orodij za razvijanje programske opreme
SVG	Scalable Vector Graphics	Stopnjevana vektorska slika
WGS84	World Geodetic System 1984	Standardiziran Zemljin koordinatni sistem
XML	eXtensible Markup Language	Razširljiv označevalni jezik

Povzetek

V diplomski nalogi sta opisana zasnova in razvoj aplikacije za mobilni operacijski sistem Android, ki uporabnikom omogoča lažje iskanje prostorov v notranjosti stavbe s prikazom tlorisa stavbe. Aplikacija je razvita v celoti s pomočjo brezplačnih ter odprtokodnih orodij in za delovanje ne potrebuje internetne povezljivosti. Prednost opisanega sistema je večja kontrola prikaza tlorisa in hitrejše posodabljanje podatkov, saj ni potrebno deliti podatkov z zunanjimi ponudniki storitev. V okviru naloge je bila aplikacija razvita na primeru nove stavbe Fakultete za računalništvo in informatiko v Ljubljani, vendar opisana rešitev služi tudi kot splošen primer razvoja aplikacij za prikazovanje notranjosti stavb. V aplikaciji lahko izbiramo med različnimi nadstropji stavbe in iščemo lokacije prostorov s pomočjo vgrajenega iskalnika. Za lažjo navigacijo v prostoru je dodan sistem prepoznavanja lokacije s pomočjo branja kod QR, ki so razporejene na različnih lokacijah v stavbi. Poleg tega aplikacija omogoča tudi prikaz najkrajše poti med dvema različnima prostoroma. Na koncu naloge je opisana tudi možnost nadgradnje aplikacije, ki bi omogočala bolj natančno prepoznavo lokacije naprave v stavbi, brez potrebe po odčitavanju kod QR.

Ključne besede: prikaz notranjosti stavbe, iskanje prostorov, mobilna aplikacija, Android, OpenStreetMap, Mapsforge

Abstract

This thesis presents the design and development of a mobile application suited for the Android operation system, which offers its users a simple way of finding locations inside a building with the use an indoor map. The application does not require internet connectivity and has been developed by means of free and open source tools. The advantage of this particular system is better control of map representation and a quicker update cycle, since there is no need to share information with external service providers. Although the application is modelled according to the plans of the new building of the Faculty of Computer and Information Science in Ljubljana, the solutions described in the thesis may be used as an example for other indoor mapping applications. The application allows the user to choose between floors of the building and find specific rooms by use of a search field. In order to simplify navigation, a location identifying system has been included by equipping several locations in the building with QR codes. In addition, the application enables the user to find the shortest path between two different locations. Finally, a possibility to upgrade the application has been offered, which would allow the location of the device to be detected more precisely within the building without having to scan the QR codes.

Keywords: indoor mapping, room location, mobile application, Android, OpenStreetMap, Mapsforge

Poglavje 1 Uvod

Prikaz zemljevida na mobilnih napravah je bil že vse od prihoda prvih pametnih telefonov ena od njihovih pomembnejših funkcij. Vsi trije večji proizvajalci mobilnih operacijskih sistemov (Google, Apple in Microsoft) ponujajo svoje samostojno razvite storitve, ki med drugim ponujajo prikaz satelitskih posnetkov, navigacijo in iskanje lokacij na zemljevidu. Z leti so aplikacije za prikaz zemljevidov razvili do te mere, da uporabnikom ponujajo celovito rešitev za navigacijo v zunanjem svetu. Nasprotno je pri prikazu in navigaciji v notranjosti stavb. Izdelava aplikacije za prikaz notranjih prostorov predstavlja večji izziv, saj je potrebno pridobiti načrt stavbe z ustreznimi dovoljenji in ga pretvoriti v obliko, ki jo lahko uporabimo v aplikaciji. Prav zaradi nedostopnosti načrtov je težje razviti aplikacijo, ki bi prikazovala notranjost poljubne stavbe.

Stavbe lahko glede na namembnost in dostopnost ločimo na javne in zasebne. Pri slednjih je odvisno od posameznega lastnika stavbe, s kom in na kakšen način bi delil podatke o notranjosti. Primer uporabe je v večjih poslovnih stavbah, kjer bi zaposlenim in gostom ponujali podatke o lokacijah posameznih pisarn in o drugih lokacijah. V tem primeru bi podatke delili le z določenimi osebami in ne s celotno javnostjo. Nasprotno je v primeru javnih zgradb, kot so na primer muzeji, knjižnice in fakultete, kjer bi te podatke lahko javno delili. V obeh primerih je potrebno ročno predelati načrt zgradbe v obliko, ki jo lahko uporabimo v aplikaciji in definirati točke zanimanja - POI (*points of interest*). V notranjosti stavbe te točke lahko prikazujejo lokacije posameznih prostorov, stopnic, dvigal, stranišč in podobno.

Aplikacija za prikaz načrta stavbe naj bi bila karseda podobna že obstoječim aplikacijam za prikaz zemljevidov. Najbolj ključen je sam prikaz načrta zgradbe, ki je v tem primeru razdeljen po nadstropjih in vključuje premikanje in povečevanje prikaza z več prstnim dotikanjem zaslona (»multitouch«). Druge pomembne funkcije so iskalnik po prostorih in pomembnih točkah, prikaz poti do določene lokacije v zgradbi ter prikaz seznama vseh lokacij. Poseben problem predstavlja določanje lokacije naprave v notranjosti. Zaradi neobstoječega ali slabega signala GPS je težje pridobiti oceno trenutne lokacije. V obstoječih raziskavah na tem področju je bilo predlaganih več rešitev tega problema [1]. Rešitev, ki je glede na težavnost in ceno izvedbe najbolj primerna, je lociranje s pomočjo vgrajenih senzorjev, ki so na voljo v današnjih mobilnih napravah [2]. Naprava odčitane meritve senzorjev primerja z že obstoječimi

meritvami, ki so bodisi shranjene v aplikaciji ali na oddaljenem strežniku. Poleg senzorjev se lahko uporabi tudi meritev jakosti signalov brezžičnih postaj, ki so v okolici naprave. Rezultat te metode pridobivanja lokacije je približna ocena z določeno mero odstopanja, ki v povprečju znaša nekaj metrov glede na dejansko lokacijo naprave [2]. Naštete metode za delovanje potrebujejo dovolj zbranih podatkov meritev brezžičnih signalov in vgrajenih senzorjev, ki jih beležimo med uporabo aplikacije. Zaradi tega smo v diplomski nalogi implementirali robustnejšo alternativo določanja lokacije s pomočjo kod QR, ki so razporejene po prostoru.

V diplomski nalogi smo predstavili razvoj aplikacije za mobilne naprave z operacijskim sistemom Android, ki ustreza prej omenjenim zahtevam. Aplikacija je v celoti razvita s pomočjo odprtokodnih rešitev in ne potrebuje zunanjih storitev ostalih ponudnikov za normalno delovanje. Poleg tega za delovanje ne potrebuje omrežne povezljivosti saj so vsi podatki shranjeni v napravi ob namestitvi aplikacije. Za praktični primer, ki je bil izdelan v okviru diplomske naloge, smo uporabili načrt zgradbe Fakultete za računalništvo in informatiko na Večni poti 133 v Ljubljani. Vendar je opisana rešitev uporabna pri razvoju kakršnekoli mobilne aplikacije za Android, ki vsebuje funkcijo prikazovanja notranjosti stavbe.

Diplomska naloga ima naslednjo strukturo. V drugem poglavju smo najprej predstavili obstoječe rešitve na tem področju in jih primerjali med seboj. V tretjem poglavju smo našteli vsa uporabljena orodja in tehnologije, ki so bile potrebne za uspešen razvoj. V četrtem poglavju smo opisali postopek priprave podatkovnega modela in tlorisa, ki smo ga uporabili v aplikaciji. Sledi peto poglavje, v katerem je opisan postopek razvoja na platformi Android s pomočjo opisanih orodij za razvoj in obstoječih programskih knjižnic. V šestem poglavju smo opisali, kako bi se lahko v bodoče izboljšalo funkcionalnost aplikacije z implementacijo sistema, ki bi natančno določal lokacijo uporabnika v prostoru. V zaključku naloge smo opisali sklepne ugotovitve. Opisali smo, v kolikšni meri je aplikacija rešila zastavljen problem in kako se primerja z že obstoječimi rešitvami.

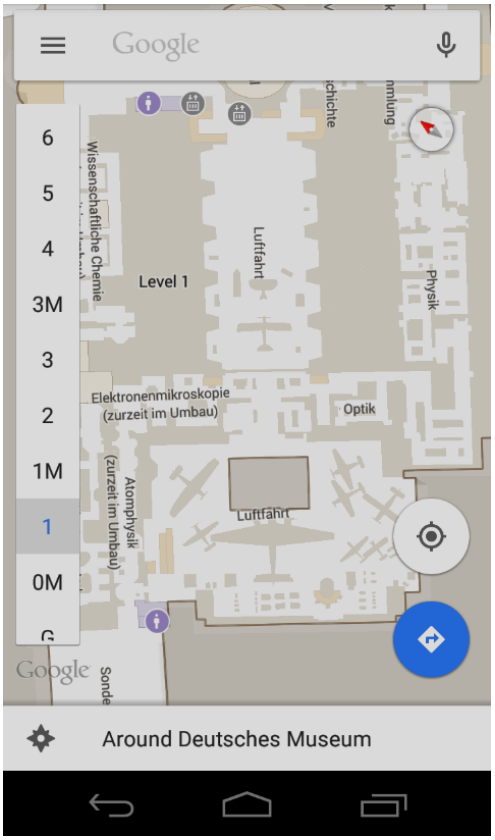
Poglavje 2 Pregled obstoječih rešitev za prikaz notranjosti stavbe

2.1 Google Indoor Maps

Brezplačna aplikacija Google Maps je na platformi Android na voljo že od prihoda prvih naprav s tem sistemom. Aplikacija je postala zelo razširjena zlasti zato, ker brezplačno ponuja vse funkcionalnosti, ki jih uporabniki potrebujejo za uspešno navigacijo v zunanjem prostoru.

Od leta 2011 ima aplikacija možnost prikaza notranjosti nekaterih javnih prostorov (Slika 2.1), kot so muzeji, knjižnice, železniške postaje, letališča itd. Notranjost stavbe se pokaže neposredno na zemljevidu ob dovolj veliki povečavi. Storitev deluje neposredno v aplikaciji, zato ima enak vmesnik kot sama aplikacija za prikaz zemljevidov. Poleg iskanja po lokacijah v notranjosti ponuja tudi izbiro nadstropja in določanje približne ocene lokacije posameznika v stavbi s pomočjo signalov brezžičnih omrežnih postaj. V času pisanja diplomske naloge storitev še ni bila na voljo na področju Slovenije.

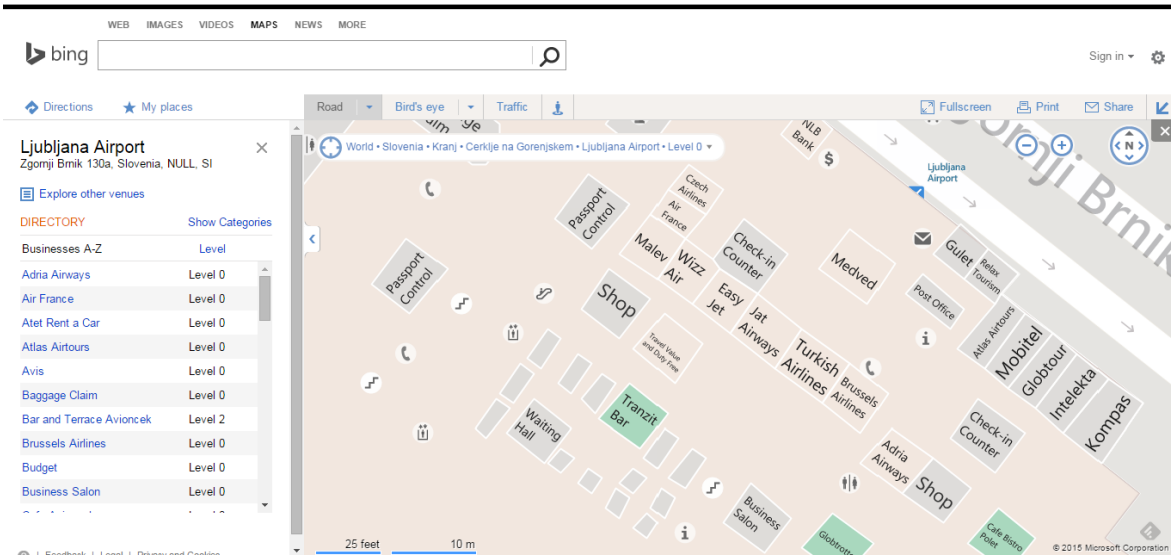
Indoor Maps je na voljo le v določenih javnih prostorih, pri katerih je Google pridobil primerne načrte stavbe. Za hitrejšo implementacijo lahko preko spletnega obrazca sami pošljemo načrt stavbe, če imamo za to ustrezna dovoljenja lastnika stavbe. Aplikacija običajno potrebuje dostop do omrežja za prenos zemljevida. Vendar obstaja možnost, kjer si uporabnik vnaprej naloži del zemljevida neposredno na napravo, da ga lahko aplikacija prikaže, ko nima več dostopa do omrežja. Načrt vsake zgradbe, ki se prikaže na zemljevidu, narišejo zaposleni pri Googlu, zato aplikacija ne omogoča samostojnega urejanja in posodabljanja podatkov.



Slika 2.1 Prikaz notranjosti tehniškega muzeja v Münchnu v aplikaciji Google Maps na Androidu

2.2 Bing Venue Maps

Podobno ima tudi Microsoftova storitev Bing Maps dodano funkcionalnost prikazovanja notranjih površin z imenom Bing Venue Maps. Storitev deluje podobno, vendar je namenska aplikacija ustvarjena samo za platformo Windows Phone in naprave s sistemom iOS. Poleg tega je na voljo preko spletnega brskalnika na računalniku. Pri tej storitvi sta slabosti potrebna omrežna povezljivost ter javni dostop vseh uporabnikov do načrta določene stavbe. Tudi ta storitev je namenjena le javnim prostorom kot v primeru Google Indoor Maps. Pri tej storitvi je sicer že na voljo nekaj lokacij v Sloveniji. To so Letališče Jožeta Pučnika (Slika 2.2) in nekaj večjih nakupovalnih središč.



Slika 2.2 Prikaz notranjosti letališča Jožeta Pučnika v spletni aplikaciji Bing Maps v brskalniku Chrome

2.3 Micello

Micello je ameriško zagonsko podjetje, ki si je za cilj izbralo ustvariti bazo načrtov notranjih prostorov po celem svetu in jih ponuditi vsem razvijalcem na mobilnih in spletnih platformah, ki jih lahko uporabijo v svojih aplikacijah. Podobno kot pri Google Indoor Maps načrt preoblikujejo in obdelajo zaposleni v podjetju, nato pa ga ponudijo vsem razvijalcem v obliki zbirke knjižnic, ki jo lahko uporabimo v aplikaciji. Pošljemo jim lahko tudi načrt stavbe v obliki slike ali CAD datoteke, če želimo določeno stavbo dodati v bazo. Platforma ima do sedaj zbranih že veliko število lokacij po vsem svetu, večinoma nakupovalna središča, posamezne trgovine ter izobraževalne ustanove.

2.4 Pomanjkljivosti obstoječih rešitev

Bistvena pomanjkljivost vseh omenjenih rešitev je odvisnost od zunanjega ponudnika, ki načrt stavbe obdela in ga ponudi v obliki, ki jo lahko uporabimo v mobilni aplikaciji. Zaradi tega te rešitve niso primerne za prikaz načrtov zasebnih prostorov, kot so na primer večje poslovne stavbe, saj so zemljevidi na voljo vsem uporabnikom storitve. Poleg tega je potrebno za vsako spremembo podatkov ali prikaza načrta, čakati na ponudnika storitve, da zemljevid posodobi. Zadnji problem obstoječih storitev je njihova dostopnost v določeni državi. V času pisanja diplomske naloge Google še ni ponujal storitve Indoor Maps v Sloveniji, Micello in Bing Venue Maps pa sta ponujala le nekaj načrtov, med katerimi so Letališče Jožeta Pučnika in nekaj večjih nakupovalnih središč. Zaradi teh pomanjkljivosti smo v diplomski nalogi predstavili izdelavo

aplikacije za prikaz notranjosti stavbe, ki omogoča samostojno posodabljanje vsebine in urejanje prikaza tlorisa.

Poglavje 3 Uporabljena tehnologija in programska orodja

3.1 Programska oprema in knjižnice

3.1.1 Android Studio in Android SDK

Android Studio je brezplačno integrirano razvojno okolje (IDE), ki ga ponuja Google za razvoj aplikacij za Android in temelji na razvojnem okolju IntelliJ IDEA. Uporaba tega razvojnega okolja je priporočljiva, saj ponuja celovito rešitev za razvoj aplikacij in ne potrebuje namestitve dodatnih vtičnikov. Pomembnejše komponente, ki jih ponuja to razvojno okolje, so:

- Urejevalnik kode, ki temelji na urejevalniku kode IntelliJ IDEA in vključuje vse funkcije tega urejevalnika (orodja za analizo in preoblikovanje kode, generiranje najbolj pogostih struktur, orodje za avtomatsko dokončanje kode glede na kontekst, v katerem se nahajamo).
- Grafični urejevalnik, s pomočjo katerega lahko v realnem času spreminjamo nastavitve grafičnih elementov v grafičnem vmesniku
- Vgrajen sistem Gradle za pomoč pri konfiguraciji in za gradnjo projekta
- Android SDK: zbirka vseh programskih knjižnic in orodij, ki jih potrebujemo za razvoj. Vključuje orodja za odpravljanje programskih napak, orodje za nalaganje in posodobitev programskih komponent ter posnemovalnik Android naprave, s katerim lahko testiramo aplikacije brez fizične naprave.

Na tem mestu lahko omenimo programsko okolje Eclipse skupaj z vtičnikom ADT – Android Development Tools, ki tudi omogoča razvoj za to platformo. Za izdelavo aplikacije smo izbrali prvo orodje, saj je specifično prilagojeno za izdelavo aplikacij za platformo Android.

3.1.2 JOSM (Java OpenStreetMap Editor)

JOSM [3] je odprtokodno grafično orodje za izdelavo zemljevidov, s katerim smo izdelali načrt stavbe v zapisu, ki smo ga nato lahko uporabili v naši aplikaciji. Orodje je eno izmed najbolj uporabljenih za izdelavo zemljevidov na platformi OSM, saj ponuja največ pripomočkov, s katerimi lažje urejamo in ustvarjamo zemljevide. Poleg tega je na voljo veliko vtičnikov, ki še

povečajo funkcionalnost orodja. Morebitna slabost tega je, da je orodje za začetnike lahko videti preveč kompleksno. Vendar je zaradi popularnosti orodja lažje najti potrebno dokumentacijo za uspešen začetek uporabe.

3.1.3 Mapsforge

Mapforge [4] je odprtokodna programska knjižnica, ki omogoča prikaz zemljevidov v aplikacijah napisanih v Javi, vključno z aplikacijami za Android. Podprte so vse različice operacijskega sistema od vključno 2.3 naprej. Prednost te knjižnice pred ostalimi je podpora za prikaz zemljevidov, ki so shranjeni neposredno na napravi, za kar ne potrebuje povezave v omrežje. Knjižnica je še v fazi razvijanja, vendar je skupnost, ki jo razvija, zelo aktivna in sproti popravlja napake in izboljšuje kodo, tako da je v trenutni fazi dovolj uporabna za naš namen. Zbirka je razdeljena na več komponent, ki so ločene v različnih paketih:

- *mapsforge-core:* splošne komponente in vmesniki
- *mapsforge-map:* splošni elementi za prikaz zemljevida
- *mapsforge-map-reader:* koda za branje zemljevidov iz datoteke
- *mapsforge-map-android:* elementi, ki se uporabljajo na platformi Android

3.1.4 Osmosis

Orodje Osmosis [5] smo uporabili za pretvorbo zapisa ».osm«, ki smo ga pridobili iz orodja JOSM v zapis ».map«, ki ga lahko uporabi knjižnica Mapsforge. Orodje je napisano v Javi in deluje v ukazni vrstici.

3.1.5 Draftsight

Draftsight [6] je zastonjski program za računalniško podprto načrtovanje (computer aided design). Je brezplačna alternativa bolj znanemu programu AutoCAD. Potrebovali smo ga za obdelavo načrta stavbe, ki smo ga prejeli v neprečiščeni obliki, z dodatnimi oznakami arhitekturnih objektov.

3.1.6 Acra (Application Crash Report for Android)

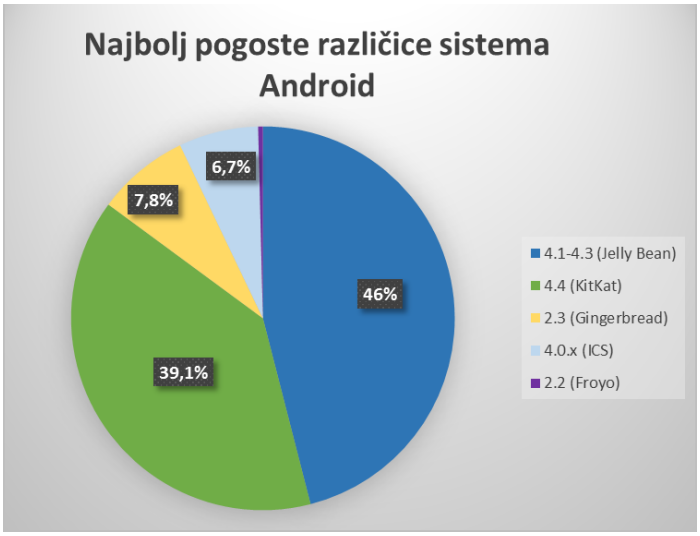
S tem orodjem lahko beležimo napake, ki se pojavijo med izvajanjem aplikacije. Orodje deluje tako, da ob vsaki napaki, zaradi katere se je program nepričakovano končal, pošlje poročilo o napaki na primerno nastavljen strežnik, kjer se shrani za nadaljnjo analizo. Takšno orodje je nepogrešljivo, če želimo beležiti napake, ko je aplikacija že izdana ali jo testira veliko testnih uporabnikov.

3.1.7 Programske knjižnice

- BarcodeScanner-Zbar: knjižnica, ki smo jo uporabili za branje QR kod, iz katerih aplikacija prepozna lokacijo naprave v stavbi.
- Android Support Library (različici v4 in v7): knjižnici za kompatibilnost s starejšimi različicami sistema. Ponujata uporabo elementov, ki so bili v razvojno platformo dodani kasneje.
- PagerSlidingTabStrip: grafični element, ki razdeli zaslon na več zavihkov, med katerimi se lahko premikamo z drsanjem po zaslonu. V aplikaciji smo ga uporabili za prikaz seznamov vseh lokacij v stavbi, razdeljenih po kategorijah.

3.2 Mobilni sistem Android

Aplikacija je bila že prvotno zamišljena za uporabo na mobilnih napravah, saj je uporabnost največja ravno takrat, ko uporabnik išče določen prostor neposredno v stavbi. Zaradi prevladujočega tržnega deleža (zlasti v Sloveniji) in enostavnejšega razvoja smo se omejil samo na operacijski sistem Android. Pri razvoju za ta sistem je opazen problem fragmentacije različic operacijskih sistemov in pripadajočih zbirk programskih knjižnic (API). Ob pregledu statistike najbolj pogosto uporabljenih različic (Slika 3.1) smo ugotovili, da naprave s sistemom Android 4.0 (API 14) in več, zavzemajo več kot 90% tržni delež [7]. V tej različici je bilo tudi veliko opaznih sprememb v arhitekturi napram prejšnjim. Kljub temu smo uporabili knjižnice za kompatibilnost, s katerimi smo zajeli tudi različice sistema Android od 2.3 (API 10) naprej. S tem smo zajeli več kot 98% naprav, ki so trenutno v uporabi na svetovnem trgu.



Slika 3.1 Diagram, ki prikazuje najbolj pogosto uporabljene različice sistema Android.

3.2.1 Arhitektura aplikacije na platformi Android

Za razvoj aplikacij na platformi Android se običajno uporablja programski jezik Java. Orodje NDK (Native development kit) sicer omogoča pisanje v jezikih C in C++, kar v določenih primerih pohitri delovanje programa. Vendar v večini primerov pohitritve niso opazne, zato se tak način razvoja ne priporoča. Aplikacije se lahko poganja na velikem številu naprav različnih oblik in velikosti. Zaradi tega je arhitektura vsake aplikacije takšna, da je logika, ki poganja aplikacijo, ločena od njene reprezentacije. Aplikacija je zgrajena iz enega ali več osnovnih gradnikov, ki imajo svojo posebno vlogo pri gradnji aplikacije [8]:

- Aktivnost (Activity): Aktivnost predstavlja okno, ki vsebuje uporabniški vmesnik aplikacije. Vsaka aplikacija, ki z uporabnikom komunicira preko vmesnika, mora implementirati vsaj eno aktivnost. Vsaka aktivnost je samostojna enota, ki lahko na različne načine komunicira z drugimi aktivnostmi in sistemom.
- Fragment: Podobno kot aktivnost se tudi s tem gradnikom tvori uporabniški vmesnik. Vendar so fragmenti namenjeni manjšim elementom uporabniškega vmesnika, ki jih obravnavamo kot isto enoto. Vsaka aktivnost ima lahko enega ali več fragmentov, ki sestavljajo celoten vmesnik.
- Storitev (Service): Storitve so komponente, ki nimajo svojega uporabniškega vmesnika in se poganjajo v ozadju sistema. Storitve se uporablja za operacije, ki potrebujejo veliko časa za dokončanje in ne potrebujejo takojšnjega odziva uporabnika. Lahko so samostojne ali pa jih zažene kakšna druga komponenta kot na primer aktivnost.
- Ponudniki vsebine (Content provider): Ta komponenta ponuja vmesnik za shranjevanje in nalaganje podatkov iz različnih virov. Je priporočljiv način za deljenje shranjenih podatkov med različnimi aplikacijami. Deluje podobno kot baza podatkov. Podatke lahko pridobimo s poizvedbami, jih spreminjamo, dodajamo ali brišemo. Edina razlika med ponudniki vsebin in bazo podatkov je v tem, da ima ponudnik vsebine več načinov shranjevanja podatkov. Lahko so shranjeni tudi v datoteki ali drugje v omrežju.
- Prejemnik obvestil (Broadcast reciever): S to komponento aplikacija beleži obvestila, ki se v sistemu pošiljajo vsem aplikacijam. Največkrat so to obvestila, ki jih proži operacijski sistem, da obvesti aplikacije o določenem dogodku, ki vpliva na stanje naprave. Sistem lahko na primer pošlje obvestilo, da se je izklopil zaslon ali o slabem stanju baterije. Obvestila lahko proži tudi določena aplikacija, če želi obvestiti ostale aplikacije o svojem stanju. To se uporablja v primerih, ko aplikacija nudi kakšno storitev, ki jo potrebujejo tudi druge aplikacije.

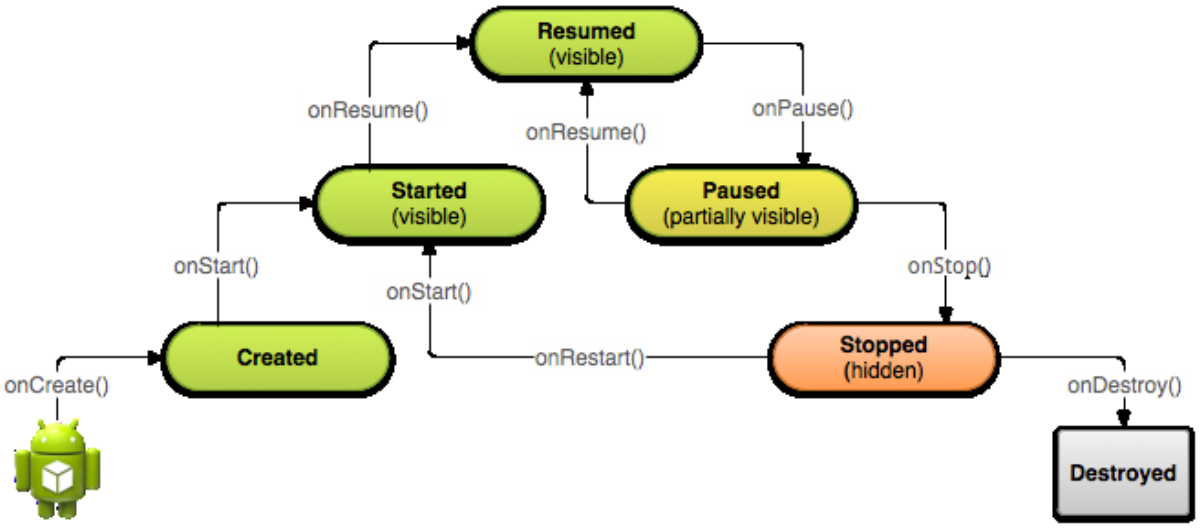
3.2.2 AndroidManifest.xml

Vsaka aplikacija za delovanje potrebuje tako imenovano »manifest« datoteko, kjer so našteti vsi osnovni gradniki aplikacije in njihove nastavitve. Pomembnejše nastavitve, ki spadajo v to datoteko, so tudi dovoljenja aplikacije za dostop do nekaterih funkcij naprave. Zaradi varnosti sistema mora vsaka aplikacija za dostop do naprednih funkcij sistema ob namestitvi od uporabnika dobiti dovoljenje za uporabo teh funkcij. Na sliki 3.2 je prikazana vsebina nastavitvene datoteke za, ki smo jo ustvarili za našo aplikacijo.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=http://schemas.android.com/apk/res/android
package="si.uni_lj.fri.friMap" android:versionCode="1"
android:versionName="1.0">

       <uses-feature android:name="android.hardware.camera" />
       <uses-permission
       android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
       <uses-permission android:name="android.permission.CAMERA" />
       <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:name="si.uni_lj.fri.friMap.MyApplication"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="si.uni_lj.fri.friMap.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <action android:name="android.intent.action.SEARCH" />
               <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name="com.dm.zbar.android.scanner.ZBarScannerActivity"
            android:label="@string/app_name"
            android:screenOrientation="landscape" />
    </application>
</manifest>
```

Slika 3.2 Vsebina nastavitvene datoteke Androidmanifest.xml

3.2.3 Življenjski cikel aktivnosti



Slika 3.3 Prikaz poteka življenjskega cikla aplikacije od trenutka, ko je ustvarjena, pa do zadnjega stanja, ko so podatki o njej uničeni.[1]

Vsak osnovni gradnik aplikacije ima svoj življenjski cikel, ki ob prehodu med stanji aplikacije proži ustrezne metode. Razložili bomo samo delovanje življenjskega cikla aktivnosti, saj smo uporabili samo ta osnovni gradnik pri razvoju naše aplikacije. Razumevanje življenjskega cikla aktivnosti je nujno, da dosežemo pravilno delovanje aplikacije [9]. Sistem lahko velikokrat nepričakovano zaustavi delovanje aplikacije in jo pošlje v ozadje. To se lahko zgodi na primer takrat, ko uporabnik med uporabo aplikacije prejme telefonski klic ali pritisne gumb za prikaz namizja. Zaradi primanjkovanja pomnilnika ali drugih sredstev lahko sistem začne izklapljati aplikacije, ki v tistem trenutku tečejo v ozadju. V vseh teh primerih se izgubijo podatki, ki jih je aplikacija v času delovanja spremenila. To pomeni, da se ob ponovnem prikazu aplikacije v ospredju prikaže prvotno stanje aplikacije. Zato sistem med izvajanjem aktivnosti, ob vsakem prehodu aktivnosti iz enega stanja v drugega, sproži ustrezne dogodke. Za lažjo predstavo je na sliki 3.3 prikazan diagram prehajanja stanj aktivnosti. Stanja, v katerih se lahko aktivnost nahaja v določenem trenutku, so sledeča:

- Ustvarjeno stanje (*Created state*): V to stanje pride aktivnost, ko je prvič ustvarjena.
- Zagnano stanje (*Started state*): To je vmesno stanje, ko pride aktivnost iz prejšnjega ustvarjenega stanja in v hitrem času preide v tekoče stanje.
- Tekoče stanje (*Running state*): Aktivnost se nahaja v tem stanju takrat, ko se nahaja v ospredju in je omogočena uporaba njenega uporabniškega vmesnika.

[1] Slika je bila objavljena na strani http://developer.android.com/training/basics/activity-lifecycle/starting.html pod licenco CC-BY-2.5.

- Prekinjeno stanje (*Paused state*): V tem stanju se aktivnost nahaja, ko je delno vidna na zaslonu in se ne odziva na uporabnikove ukaze. V ospredju je druga aktivnost, ki delno prekriva zaslon.
- Ustavljeno stanje (*Stopped state*): V tem stanju aktivnost ni več vidna na zaslonu. Lahko je prekrita z drugo aktivnostjo ali začasno ustavljena. V tem stanju so še shranjeni podatki o aktivnosti vendar se koda ne izvaja več.
- Uničena aktivnost (*Destroyed activity*): Aktivnost se dokončno uniči, ko sistemu zmanjka razpoložljivih sredstev za poganjanje tekočih aktivnosti. Aktivnost se lahko uniči le, če se je prej nahajala v prekinjenih ali ustavljenih stanjih.

Vsaka aktivnost deduje metode razreda Activity, ki jih sistem sproži ob prehodu med stanji aktivnosti. V primeru, če želimo aktivirati kodo, ki se proži ob določenem dogodku, moramo v implementaciji naše aktivnosti prepisati primerne metode:

- *onCreate():* Ta dogodek se sproži ob prvem zagonu aktivnosti. V tej metodi lahko nastavimo vse spremenljivke in povežemo podatke s seznami.
- *onResume():* Ta dogodek se sproži, ko aktivnost postane vidna in se odziva na ukaze uporabnika. V tej metodi se lahko ponovno vzpostavi prejšnje stanje, če je bila aktivnost začasno ustavljena. Prav tako lahko na tem mestu zaženemo vse animacije in poslušalce dogodkov.
- *onPause():* Ta dogodek se sproži tik preden se aktivnost ustavi. V tej metodi moramo shraniti stanje aktivnosti, saj se lahko aktivnost v ozadju izbriše iz pomnilnika. Priporočljivo je, da na tem mestu ustavimo vse animacije in poslušalce dogodkov. Pomembno je, da se koda v tej metodi izvede hitro, saj mora nova aktivnost čakati na dokončanje te metode preden se lahko uspešno zažene.
- *onStop():* Ta dogodek se sproži, ko preide aktivnost v ozadje. V tej metodi lahko sprostimo sredstva, ki jih je uporabljala aktivnost.
- *onDestroy():* Zadnji dogodek, ki se sproži tik preden sistem uniči instanco aktivnosti, da privarčuje s prostorom pomnilnika. Velikokrat ne potrebujemo pisati kode v tej metodi, saj zadostujeta metodi *onPause()* in *onStop()*. To metodo kličemo v primeru, če moramo sprostiti še zadnje vire, ki jih je uporabljala aktivnost. Pomembno je, da se konča vse dodatne procese, ki jih je potrebovala aktivnost za normalno delovanje.

3.2.4 Fragmenti

Fragmenti so dodaten gradnik za uporabo pri razvoju aplikacije, ki so se v sistemu Android pojavili z različico 3.0. Osnovni namen fragmentov je omogočanje prilagoditve uporabniškega

vmesnika različnim napravam. Vsak fragment je samostojna enota, ki jo lahko poljubno in dinamično vstavljamo določeni aktivnosti. Fragmentov tako ne moremo uporabljati brez aktivnosti, na katero so neposredno vezani. Uporabimo jih lahko tudi brez vmesnika, tako da končnemu uporabniku niso vidni. V tem primeru jih lahko uporabimo za implementacijo kode, ki je modularna in uporabna na več mestih aplikacije. Fragmenti imajo svoj življenjski cikel, podobno kot aktivnost, vendar se od njih malo razlikuje. Dodaten dogodek, ki se sproži v trenutku, ko mora fragment izrisati svoj uporabniški vmesnik, je *onCreateView()*. Nasproten dogodek temu je *onDestroyView()*, ki se sproži, ko gre fragment v ozadje.

3.3 Open Street Map

Glede na zastavljene cilje in željo po uporabi odprtokodnih rešitev, smo za prikaz načrta stavbe uporabili zapis OpenStreetMap (OSM), ki je na voljo pod odprto licenco Creative Commons. OSM je odprta platforma, ki uporablja množični model za ustvarjanje svetovnega zemljevida. Pri tem ima lahko vsak posameznik možnost izboljšave zemljevida s pomočjo odprtokodnih orodij. Prav zaradi velikega števila uporabnikov, ki prostovoljno popravljajo in dodajajo kartografske elemente, se je kvaliteta zemljevidov čez leta izboljšala do te mere, da se lahko primerja oziroma v nekaterih pogledih celo preseže zaprte platforme. Zaradi svoje odprte narave je tehnologijo možno uporabiti tudi za ustvarjanje lastnih zemljevidov s svojo grafično podobo.

Tudi Google ponuja zbirko orodij Google Maps API, s katero lahko aplikaciji dodamo prikaz zemljevida. Poleg prikaza zemljevida z vsemi funkcijami, ki so prisotne v aplikaciji Google Maps, lahko na zemljevidu prikažemo dodatne sloje z elementi, kot so poti, ikone in slike. Kljub temu smo se odločili za odprto platformo OpenStreetMap, saj ponuja več svobode pri nastavitvi prikaza kartografskih elementov. Poleg tega smo za prikaz načrta stavbe uporabili knjižnico Mapsforge, ki omogoča branje podatkov neposredno iz naprave.

3.3.1 OSM XML

Večina orodij na platformi OSM uporablja format XML za zapis podatkov o zemljevidih [10]. Vsaka datoteka v OSM XML formatu ima določeno strukturo. Zapis se začne s predpono *<?xml>,* ki definira različico XML formata in kodiranje znakov. Nato sledi korenski element *<osm>,* pri katerem v atributu navedemo različico OSM formata in program, s pomočjo katerega smo ustvarili dokument. Za ponazoritev kartografskih elementov imamo na voljo štiri osnovne gradnike. To so vozlišča, poti, relacije in oznake.

Vozlišče (<node>):

Vozlišče predstavlja edinstveno lokacijo, ki se nahaja na zemljevidu sveta. Vsako vozlišče mora imeti podan atribut za identifikacijo ter koordinate v standardu WGS84. Vozlišče lahko predstavlja samostojno lokacijo, lahko pa je sestavni del poti ali relacije.

Pot (<way>):

Vsebuje več vozlišč. Pot je lahko odprta ali zaprta. Slednja ima enako prvo in zadnje vozlišče. Zaprte poti se uporabljajo za ponazoritev objektov ali površin. Odprte poti se uporablja za ponazoritev poti, cest, rek in podobno.

Relacija(<relation>):

Vsebuje vozlišča in poti, ki jih povezujemo v isti kontekst. S tem elemente grupiramo v skupne enote, ki imajo določene lastnosti. Pomen relacije je določen s pripadajočimi oznakami, ki jih vsebuje. Pogost primer relacije, ki smo jo uporabili tudi v naši aplikaciji, je »multipolygon«. S tem ponazorimo površino, ki vsebuje prazen prostor.

Oznaka(<tag>):

Oznake se uporabljajo za določanje pomena osnovnih elementov. Sestavljene so iz ključa in pripadajoče vrednosti. Vozlišča, poti in relacije imajo lahko eno ali več oznak. Oznake niso nikjer točno definirane, vendar se skupnost drži splošnih dogovorov za uporabo najbolj pogostih oznak. Razvijalec zemljevida ima možnost slediti dogovorom ali uporabiti novo oznako, če ni obstoječe primerne oznake. Na sliki 3.4 je prikazan primer zapisa datoteke OSM XML, ki vsebuje nekaj vozlišč in pot.

```
1  <osm generator="JOSM" upload="true" version="0.6">
2      <node action="modify" changeset="1" id="-560" lat="46.04937099663" lon="14.468739041" timestamp=
       "2015-01-28T13:38:04Z" version="1" visible="true" />
3      <node action="modify" changeset="1" id="-2308" lat="46.04935874076" lon="14.46873904739" timestamp=
       "2015-01-28T13:38:04Z" version="1" visible="true" />
4      <node action="modify" changeset="1" id="-10" lat="46.0493587406" lon="14.46873841218" timestamp=
       "2015-01-28T13:38:04Z" version="1" visible="true" />
5      <node action="modify" changeset="1" id="-8" lat="46.04937099647" lon="14.46873840579" timestamp=
       "2015-01-28T13:38:04Z" version="1" visible="true" />
6      <node action="modify" changeset="1" id="-560" lat="46.04937099663" lon="14.468739041" timestamp=
       "2015-01-28T13:38:04Z" version="1" visible="true" />
7
8      <way action="modify" changeset="1" id="-3688" timestamp="2015-01-28T13:38:04Z" version="1" visible="true">
9          <nd ref="-560" />
10         <nd ref="-2308" />
11         <nd ref="-10" />
12         <nd ref="-8" />
13         <nd ref="-560" />
14         <tag k="door" v="yes" />
15     </way>
16 </osm>
```

Slika 3.4 Primer OSM XML zapisa, ki vsebuje nekaj vozlišč (node) in pot (way)

3.3.2 Oznake za prikaz notranjih prostorov

Do sedaj skupnost OSM še ni prišla do dogovora glede uporabe oznak za prikaz notranjih prostorov. Obstajajo samo priporočila in osnutki [11], ki nam služijo kot pomoč pri izbiri oznak. Pri oblikovanju prikaza stavbe v naši aplikaciji smo tudi mi sledili tem priporočilom. Sicer ni pomembno, kakšne oznake izberemo, ker so podatki shranjeni neposredno v aplikaciji. Vendar je bolje, da je sistem zgrajen tako, da kasneje omogoča lažje prilagajanje in združljivost s celotno platformo.

V predlagani označevalni shemi [12], ki smo jo uporabili tudi v naši aplikaciji, so elementi ločeni po nadstropjih. Vsakemu elementu, ki se nahaja v stavbi je prirejena oznaka, ki ima določeno številko nadstropja. Posebni elementi, ki se nahajajo v notranjosti, imajo tudi določeno svojo oznako. Tako lahko definiramo, kje se nahajajo stopnice, dvigala, zidovi in ostalo. Za izdelavo načrta stavbe, uporabljenega v aplikaciji, smo določili naslednje oznake:

- indoor=door: Prikaz vrat
- indoor=wall: Prikaz zidov
- indoor=window: Prikaz oken
- stairs=yes: Prikaz stopnic
- highway=elevator: Prikaz dvigala

Poglavje 4 Zasnova mobilne aplikacije »FRI map«

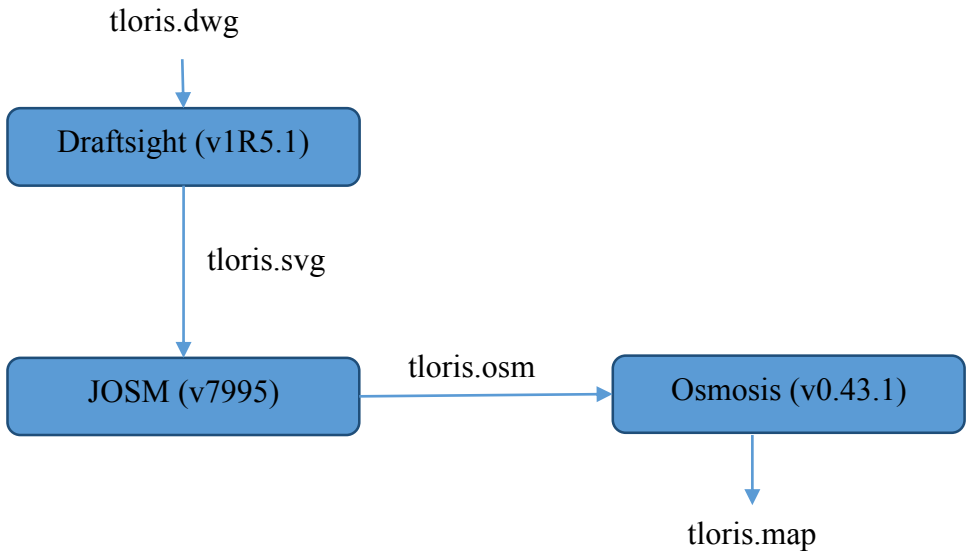
4.1 Cilji in opis aplikacije

Osnovni namen aplikacije, ki smo jo razvili v sklopu diplomske naloge, je, da uporabnikom olajša iskanje prostorov in orientacijo v notranjosti določene stavbe. Stavba, na podlagi katere smo izdelali aplikacijo, je Fakulteta za računalništvo in informatiko v Ljubljani. Aplikacija je še posebno primerna za študente, ki potrebujejo informacijo o lokaciji kabinetov določenih profesorjev ali laboratorijev. Drug primer uporabe je pri raznih gostih fakultete, ki jim lokacije prostorov še niso znane.

Aplikacija ima naslednje funkcije:

- Prikaz tlorisa stavbe z imeni prostorov in ikonami, ki označujejo točke zanimanja POI (*point of interest*). Prikaz je razdeljen po nadstropjih stavbe in omogoča premikanje ter povečevanje tlorisa.
- Iskalno polje, s katerim lahko poiščemo prostor ali osebo, ki se nahaja v prostoru.
- Izbira prostora iz seznama, ki je razdeljen na različne kategorije (predavalnice, laboratoriji, kabineti in ostalo).
- Prikaz poti od začetne do končne lokacije na prikazovalniku načrta stavbe.
- Določanje in prikaz trenutne lokacije uporabnika v stavbi s pomočjo kode QR, ki jo lahko uporabnik odčita na določenih lokacijah v stavbi.

4.2 Izdelava tlorisa stavbe za uporabo v aplikaciji



Slika 4.1 Diagram prikazuje uporabljena orodja za pretvorbo tlorisa.

Za izris tlorisa stavbe smo morali najprej ustrezno preoblikovati zapis tlorisa tako, da smo ga lahko uporabili v aplikaciji. Slika 4.1 prikazuje celoten postopek pretvorbe zapisa z vsemi programi, ki smo jih potrebovali in vhodnimi ter izhodnimi datotekami, v katerih so bili shranjeni tlorisi posameznih nadstropij.

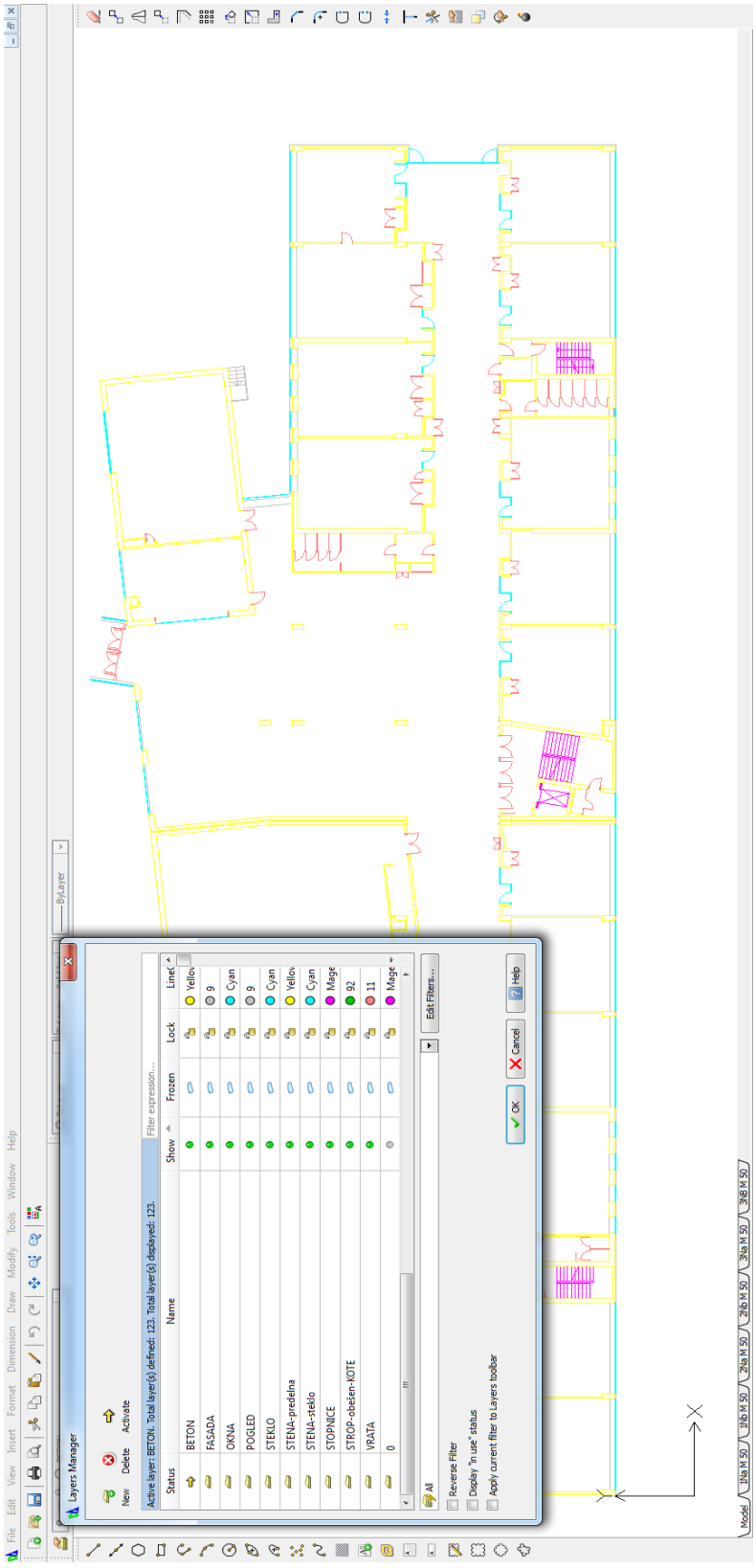
Nova stavba fakultete je bila dokončana leta 2014. Zaradi tega smo lahko pridobili zelo natančne in ažurne arhitekturne podatke, ki so nam koristili pri izdelavi poenostavljenega tlorisa. Načrt stavbe smo prejeli v binarnem formatu DWG, ki se uporablja za shranjevanje podatkov v programih za računalniško podprto načrtovanje. Za obdelavo načrtov smo uporabili brezplačni program Draftsight, ki je za naše potrebe zadostoval. Načrt je bil v datoteki razdeljen po nadstropjih in je vseboval veliko dodatnih informacij, ki so naredile tloris manj razpoznaven. Informacije o arhitekturnih elementih so bile razdeljene po slojih, ki se jih je dalo po želji izklopiti, tako da smo lahko ustvarili bolj pregleden tloris. Sloji so bili ustrezno poimenovani, zato smo lahko enostavno izločili nepotrebne elemente.

Poenostavljen načrt je vseboval le naslednje osnovne elemente:

- Zidovi
- Vrata
- Okna
- Fasada stavbe
- Stopnišča
- Dvigalo

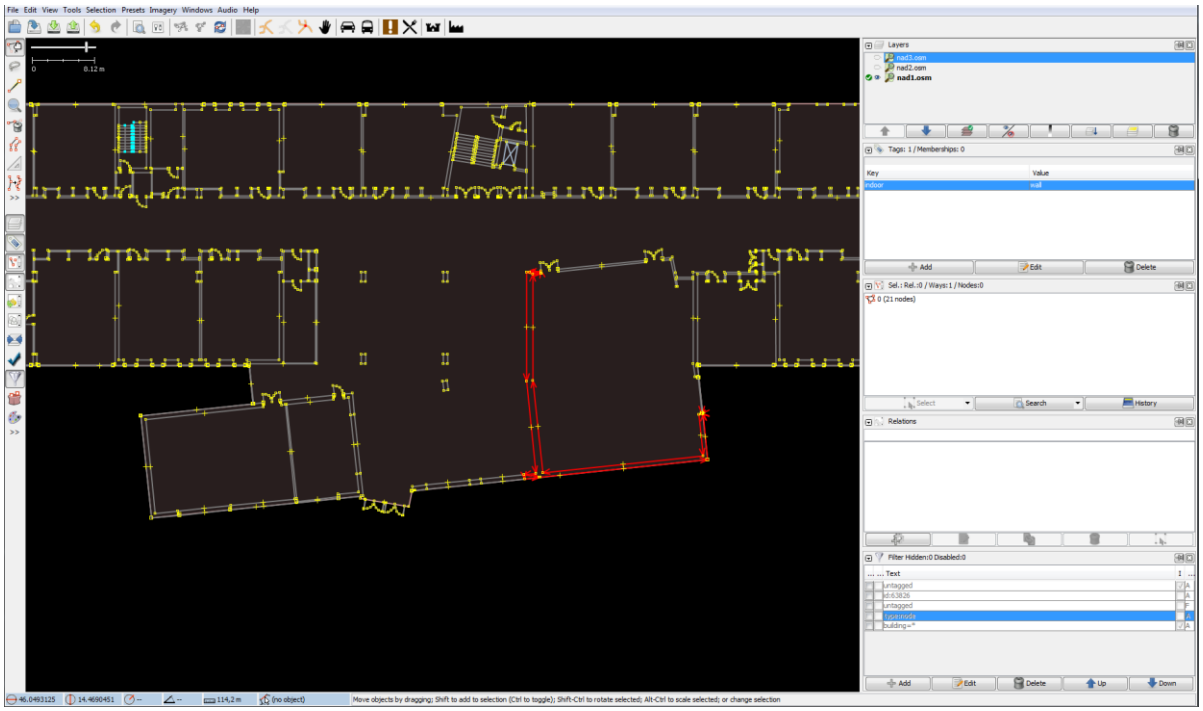
Slika 4.2 prikazuje prvotno neprečiščeno obliko načrta, medtem ko je na sliki 4.3 prikazana prečiščena različica. Poleg tega so bile priročne tudi enolične oznake prostorov, ki smo jih kasneje uporabili pri identifikaciji prostorov v podatkovni plasti aplikacije. Načrt smo nato izvozili v vektorskem zapisu SVG za naknadno obdelavo. Pri tem smo ločili nadstropja po posameznih datotekah.

Slika 4.2 Prikaz načrta prvega nadstropja fakultete v prvotni neprečiščeni obliki



Slika 4.3 S programom Draftsight smo lahko preuredili prikaz načrta (na prejšnji sliki) v bolj enostavno in pregledno obliko

Naslednji korak obdelave tlorisa je potekal v programu JOSM (Java OpenStreetMap Editor), ki nam omogoča obdelavo zemljevidov v zapisu, ki se uporablja na platformi OpenStreetMap. Vir podatkov za obdelavo lahko prenesemo iz prosto dostopnih strežnikov, ki gostijo zemljevid celotnega sveta. S programom lahko prenesemo katerokoli območje na svetu in po želji uredimo ali dodajamo elemente. Spremembe nato pošljemo nazaj na strežnik, ki jih ustrezno shrani. Druga možnost, ki smo jo uporabili v našem primeru, je ročna izdelava zemljevida s pomočjo podatkov shranjenih lokalno na računalniku. Za pomoč pri izdelavi lahko uporabimo sliko tlorisa, ki je v stavbah pogostokrat na voljo, saj ima večina javnih prostorov evakuacijske načrte postavljene na vidnih mestih. Podatke se lahko uvozi tudi neposredno iz ene od podprtih datotek. V našem primeru so bile datoteke v vektorskem zapisu SVG.



Slika 4.4 Programsko orodje JOSM, s katerim smo izdelali tloris za uporabo v aplikaciji

Vmesnik programa JOSM vsebuje urejevalnik, kjer lahko označimo vozlišča ali poti in jim določimo oznake ali spremenimo njihovo postavitev na zemljevidu. Na sliki 4.4 je prikazano delovanje programa med urejanjem zemljevida. Urejevalnik predstavlja območje na zemljevidu sveta. Vsakemu vozlišču, ki ga vstavimo v urejevalnik, se samodejno določijo koordinate, ki so odvisne od tega, kam na zemljevidu postavimo vozlišče. Ob uvozu vektorske datoteke v program je bil tloris ponazorjen s točkami in daljicami, brez koristnih informacij, v kakšni obliki naj se elementi prikažejo na zemljevidu. Zato smo morali označiti vse točke in povezave med njimi in jim določiti ustrezne oznake OSM, ki smo jih opisali v poglavju 3.3.2.

Zadnji korak obdelave je bil pretvorba v binarni zapis .map, ki se uporablja za prikaz zemljevida s knjižnico Mapsforge. V ta namen smo uporabili orodje Osmosis. Orodje je namenjeno branju, zapisovanju in analiziranju podatkov, ki so shranjeni v formatu OSM. Napisano je v programskem jeziku Java in se zažene preko ukazne vrstice. Orodje je bilo napisano tako, da se lahko poljubno razširi njegovo funkcionalnost z vtičniki drugih razvijalcev. V knjižnici Mapsforge je priložen vtičnik *mapsforge-writer*, ki smo ga potrebovali za ustrezno pretvorbo datoteke.

Poleg zemljevida smo morali priložiti nastavitveno datoteko *tags-mapping.xml* (Slika 4.5). To datoteko smo potrebovali, saj smo dodali nove oznake elementov, ki še niso ustrezno zapisani v standardu OSM, zato jih orodje za pretvorbo ni samodejno prepoznalo. V datoteki lahko določimo, katere oznake se bodo prikazale na zemljevidu in pod kakšno povečavo zemljevida se bodo objekti začeli prikazovati, ne moremo pa definirati, v kakšni obliki se objekti prikažejo To se nastavi naknadno v aplikaciji z nastavitveno datoteko renderheme.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<tag-mapping xmlns=http://mapsforge.org/tag-mapping
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://mapsforge.org/tag-mapping ../resources/tag-
mapping.xsd" default-zoom-appear="16" profile-name="default-profile">
      <ways>
            <osm-tag key="indoor" value="door" zoom-appear="14" />
            <osm-tag key="stairs" value="yes" zoom-appear="14" />
            <osm-tag key="highway" value="elevator" zoom-appear="14" />
            <osm-tag key="indoor" value="wall" zoom-appear="14" />
            <osm-tag key="indoor" value="window" zoom-appear="14" />
      </ways>
</tag-mapping>
```

Slika 4.5 Ukazi dodatnih elementov, ki smo jih uporabili za prikaz tlorisa

Za pretvorbo iz OSM formata v binarno datoteko .map smo uporabili ukaz:

osmosis --read-xml file=nad1_attr.osm --mapfile-writer file=nad1.map tag-conf-file=tag-mapping.xml bbox=46.0490820,14.4681220,46.0495640,14.4700700

Prvi argument je naslov vhodne datoteke, drugi je naslov izhodne datoteke, tretji je naslov nastavitvene datoteke *tag-mapping.xml* in zadnji argument predstavlja koordinate pravokotnika, ki opredeljuje območje, kjer se nahaja shranjen zemljevid.

4.3 Podatkovni model prostorov in osebja

Poleg samega prikaza tlorisa je funkcija naše aplikacije tudi iskanje določenega prostora v stavbi in prikaz dodatnih informacij prostorov na prikazovalniku načrta stavbe. Zato smo morali

določiti ustrezen podatkovni model, ki nam je omogočal lažji prikaz in iskanje po podatkih. Model smo zasnovali tako, da bo kasneje omogočal enostavno dodajanje drugih tipov prostorov. Prostore smo najprej, glede na namen uporabe, razdelili v različne kategorije. Prostori, ki jih lahko najdemo v stavbi fakultete, so predavalnice, laboratoriji, kabineti, pisarne in ostali splošno namenski prostori.

Najprej smo določili osnovni tip prostora z atributi, ki si jih delijo vsi ostali prostori. Vsi prostori, ki se nahajajo v stavbi, tako dedujejo atribute od tega osnovnega tipa in nato dodatno definirajo svoje. Vsak prostor ima določen identifikator, kratico in ime prostora. Laboratoriji, kabineti in pisarne imajo določen tudi seznam oseb, ki prostor uporabljajo. V primeru kabinetov in pisarn ima lahko ena oseba samo en prostor, v primeru laboratorijev pa te omejitve ni.

Drugi osnovni tip predstavlja osebje fakultete. V ta tip spadajo vse osebe, ki imajo določen prostor na fakulteti. Atributi, ki jih ima ta tip, so ime, priimek ter identifikator prostora, ki povezuje osebo s prostorom.

Za vir podatkov smo uporabili Excel razpredelnico, ki smo jo pridobili v tajništvu fakultete. Prostori so bili razdeljeni v dva zavihka, ki sta vsebovala podatke o kabinetih in laboratorijih. Razpredelnici sta vsebovali tudi nekaj dodatnih podatkov, ki jih nismo potrebovali. Za lažje branje podatkov v aplikaciji smo ustvarili nove razpredelnice z izluščenimi podatki in jih shranili v zapisu CSV (*comma separated values*). Za zapis CSV smo se odločili, saj omogoča enostavno branje v aplikaciji in hkrati omogoča enostavno ročno urejanje podatkov. V tem zapisu so stolpci razpredelnice ločeni z določenim znakom (v našem primeru je bil ta znak podpičje). Program lahko enostavno prebere datoteko tako, da najprej prebere vsako vrstico posebej, nato se lahko podatke v stolpcih loči z uporabo regularnega izraza.

Ustvarili smo tri razpredelnice s podatki. V prvi smo določili vse prostore na fakulteti (Slika 4.6). Stolpci, ki jih vsebuje razpredelnica, so:

- Šifra: Oznaka, ki enolično določa prostor v stavbi. Šifro smo pridobili iz načrta stavbe. Ima format RX.XX. Prva številka določa nadstropje v katerem se prostor nahaja, drugi dve pa določata številko prostora v nadstropju. Ta stolpec je nujen za prepoznavo prostora.
- Tip: Prostore smo ločili na več kategorij. To so predavalnice, kabineti, laboratoriji in ostalo (dekanat, stopnišča, dvigalo, stranišča). Tudi ta stolpec je nujen za prepoznavo.
- Ime: Dolgo ime prostora. Pri kabinetih ta zapis ni nujen, saj v tem kontekstu nima pomena.

- Kratica: Kratica prostora, ki jo potrebujemo za oznako prostora na prikazu načrta stavbe. Potrebna je pri predavalnicah in laboratorijih. Vendar je pri laboratorijih potrebna samo, če je definirano tudi dolgo ime.

V drugi razpredelnici so zapisani podatki o vseh osebah, ki imajo določen prostor (Slika 4.7). Vsebuje naslednje stolpce:

- Ime osebe
- Priimek osebe
- Kabinet: Šifra prostora, ki se mora ujemati s šifro v prejšnji razpredelnici.
- Laboratorij: Kratica laboratorija, v katerem se nahaja oseba. Ta zapis ni nujen.

V tretji razpredelnici so zapisani podatki o vseh laboratorijih (Slika 4.8). Vsebuje naslednje stolpce:

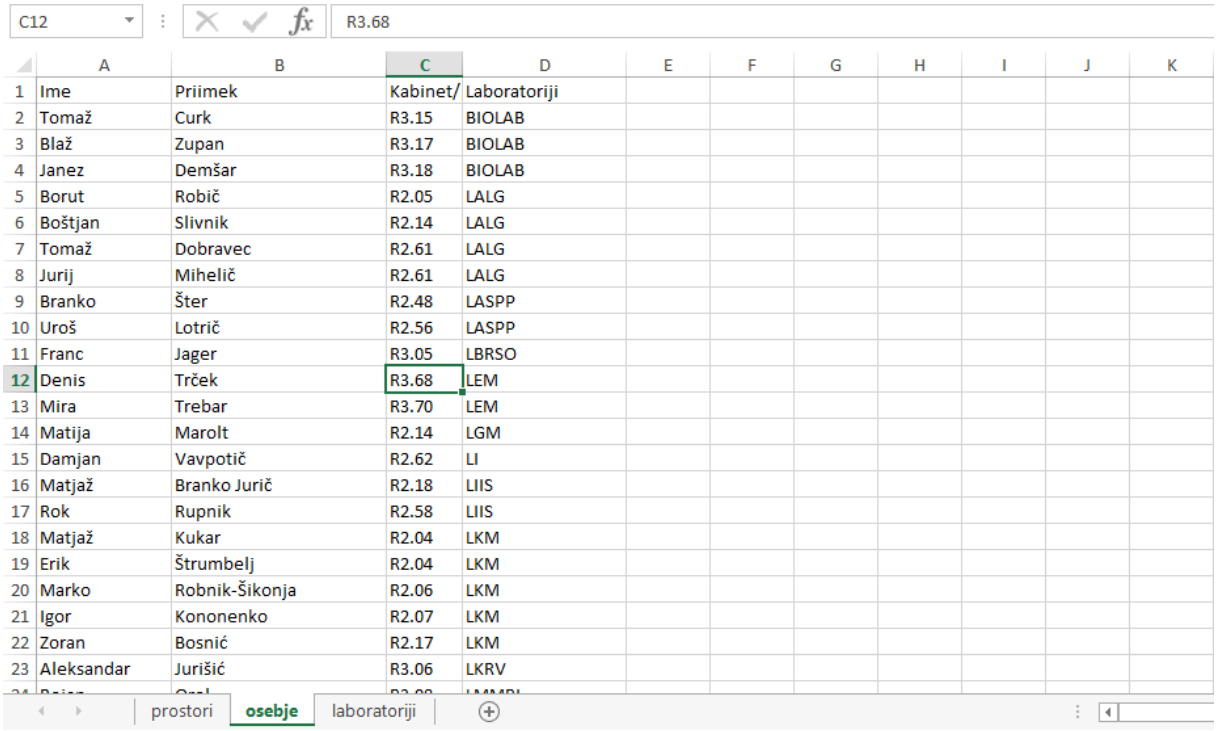
- Kratica: Identifikator laboratorija. Podatek mora biti enak kot pri prejšnjih dveh razpredelnicah, saj enolično določa laboratorij. V tem primeru nismo uporabili šifre prostora, saj ima lahko laboratorij več prostorov. Poleg tega lahko laboratorij v prihodnosti zamenja prostor, v katerem se nahaja.
- Vodja laboratorija
- Zaposleni: Seznam oseb, ki so zaposlene v laboratoriju. Zapisi so ločeni z vejico.

A2 | R1.23

	A	B	C	D	E	F	G	H	I	J	K
1	Šifra	Tip	Ime	Kratica							
2	R1.23	Predavalnica	Predavalnica 1	P01							
3	R1.25	Predavalnica	Predavalnica 2	P02							
4	R1.27	Predavalnica	Predavalnica 3	P03							
5	R1.28	Predavalnica	Predavalnica 4	P04							
6	R1.29	Predavalnica	Predavalnica 5	P05							
7	R1.30	Predavalnica	Predavalnica 6	P06							
8	R1.32	Predavalnica	Predavalnica 7	P07							
9	R1.33	Predavalnica	Predavalnica 8	P08							
10	R1.38	Predavalnica	Predavalnica 9	P09							
11	R1.39	Predavalnica	Predavalnica 10	P10							
12	R1.40	Predavalnica	Predavalnica 11	P11							
13	R1.41	Predavalnica	Predavalnica 12	P12							
14	R1.04	Predavalnica	Predavalnica 14	P14							
15	R1.05	Predavalnica	Predavalnica 15	P15							
16	R1.10	Predavalnica	Predavalnica 16	P16							
17	R1.11	Predavalnica	Predavalnica 17	P17							
18	R1.13	Predavalnica	Predavalnica 18	P18							
19	R1.14	Predavalnica	Predavalnica 19	P19							
20	R1.15	Predavalnica	Predavalnica 20	P20							
21	R1.16	Predavalnica	Predavalnica 21	P21							
22	R1.19	Predavalnica	Predavalnica 22	P22							
23	R1.20	Ostalo	Diplomska soba								

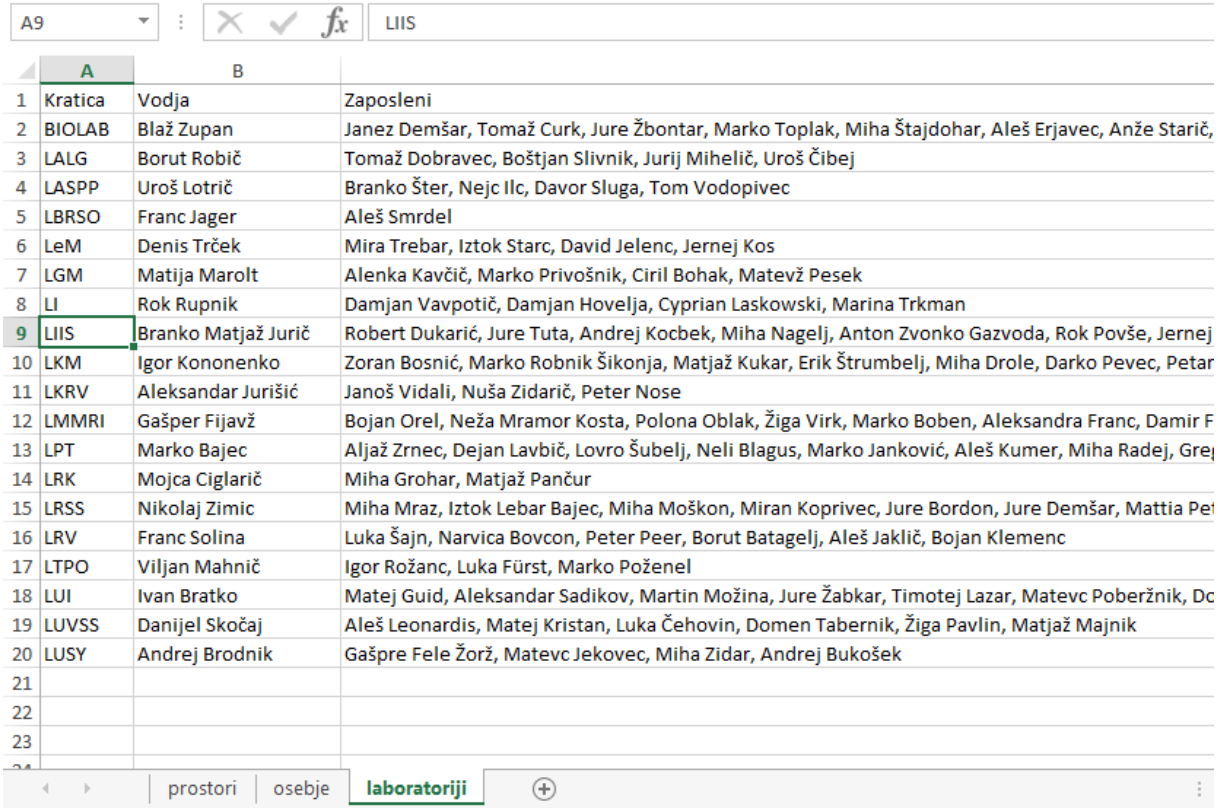
prostori | osebje | laboratoriji

Slika 4.6 Razpredelnica s podatki o prostorih



	A	B	C	D	E	F	G	H	I	J	K
1	Ime	Priimek	Kabinet/	Laboratoriji							
2	Tomaž	Curk	R3.15	BIOLAB							
3	Blaž	Zupan	R3.17	BIOLAB							
4	Janez	Demšar	R3.18	BIOLAB							
5	Borut	Robič	R2.05	LALG							
6	Boštjan	Slivnik	R2.14	LALG							
7	Tomaž	Dobravec	R2.61	LALG							
8	Jurij	Mihelič	R2.61	LALG							
9	Branko	Šter	R2.48	LASPP							
10	Uroš	Lotrič	R2.56	LASPP							
11	Franc	Jager	R3.05	LBRSO							
12	Denis	Trček	R3.68	LEM							
13	Mira	Trebar	R3.70	LEM							
14	Matija	Marolt	R2.14	LGM							
15	Damjan	Vavpotič	R2.62	LI							
16	Matjaž	Branko Jurič	R2.18	LIIS							
17	Rok	Rupnik	R2.58	LIIS							
18	Matjaž	Kukar	R2.04	LKM							
19	Erik	Štrumbelj	R2.04	LKM							
20	Marko	Robnik-Šikonja	R2.06	LKM							
21	Igor	Kononenko	R2.07	LKM							
22	Zoran	Bosnić	R2.17	LKM							
23	Aleksandar	Jurišić	R3.06	LKRV							

Slika 4.7 Razpredelnica s podatki o osebah



	A	B	
1	Kratica	Vodja	Zaposleni
2	BIOLAB	Blaž Zupan	Janez Demšar, Tomaž Curk, Jure Žbontar, Marko Toplak, Miha Štajdohar, Aleš Erjavec, Anže Starič,
3	LALG	Borut Robič	Tomaž Dobravec, Boštjan Slivnik, Jurij Mihelič, Uroš Čibej
4	LASPP	Uroš Lotrič	Branko Šter, Nejc Ilc, Davor Sluga, Tom Vodopivec
5	LBRSO	Franc Jager	Aleš Smrdel
6	LeM	Denis Trček	Mira Trebar, Iztok Starc, David Jelenc, Jernej Kos
7	LGM	Matija Marolt	Alenka Kavčič, Marko Privošnik, Ciril Bohak, Matevž Pesek
8	LI	Rok Rupnik	Damjan Vavpotič, Damjan Hovelja, Cyprian Laskowski, Marina Trkman
9	LIIS	Branko Matjaž Jurič	Robert Dukarić, Jure Tuta, Andrej Kocbek, Miha Nagelj, Anton Zvonko Gazvoda, Rok Povše, Jernej
10	LKM	Igor Kononenko	Zoran Bosnić, Marko Robnik Šikonja, Matjaž Kukar, Erik Štrumbelj, Miha Drole, Darko Pevec, Petar
11	LKRV	Aleksandar Jurišić	Janoš Vidali, Nuša Zidarič, Peter Nose
12	LMMRI	Gašper Fijavž	Bojan Orel, Neža Mramor Kosta, Polona Oblak, Žiga Virk, Marko Boben, Aleksandra Franc, Damir F
13	LPT	Marko Bajec	Aljaž Zrnec, Dejan Lavbič, Lovro Šubelj, Neli Blagus, Marko Janković, Aleš Kumer, Miha Radej, Greg
14	LRK	Mojca Ciglarič	Miha Grohar, Matjaž Pančur
15	LRSS	Nikolaj Zimic	Miha Mraz, Iztok Lebar Bajec, Miha Moškon, Miran Koprivec, Jure Bordon, Jure Demšar, Mattia Pet
16	LRV	Franc Solina	Luka Šajn, Narvica Bovcon, Peter Peer, Borut Batagelj, Aleš Jaklič, Bojan Klemenc
17	LTPO	Viljan Mahnič	Igor Rožanc, Luka Fürst, Marko Poženel
18	LUI	Ivan Bratko	Matej Guid, Aleksandar Sadikov, Martin Možina, Jure Žabkar, Timotej Lazar, Matevc Poberžnik, Dc
19	LUVSS	Danijel Skočaj	Aleš Leonardis, Matej Kristan, Luka Čehovin, Domen Tabernik, Žiga Pavlin, Matjaž Majnik
20	LUSY	Andrej Brodnik	Gašpre Fele Žorž, Matevc Jekovec, Miha Zidar, Andrej Bukošek
21			
22			
23			

Slika 4.8 Razpredelnica s podatki o laboratorijih

4.4 Izdelava mreže povezav med prostori

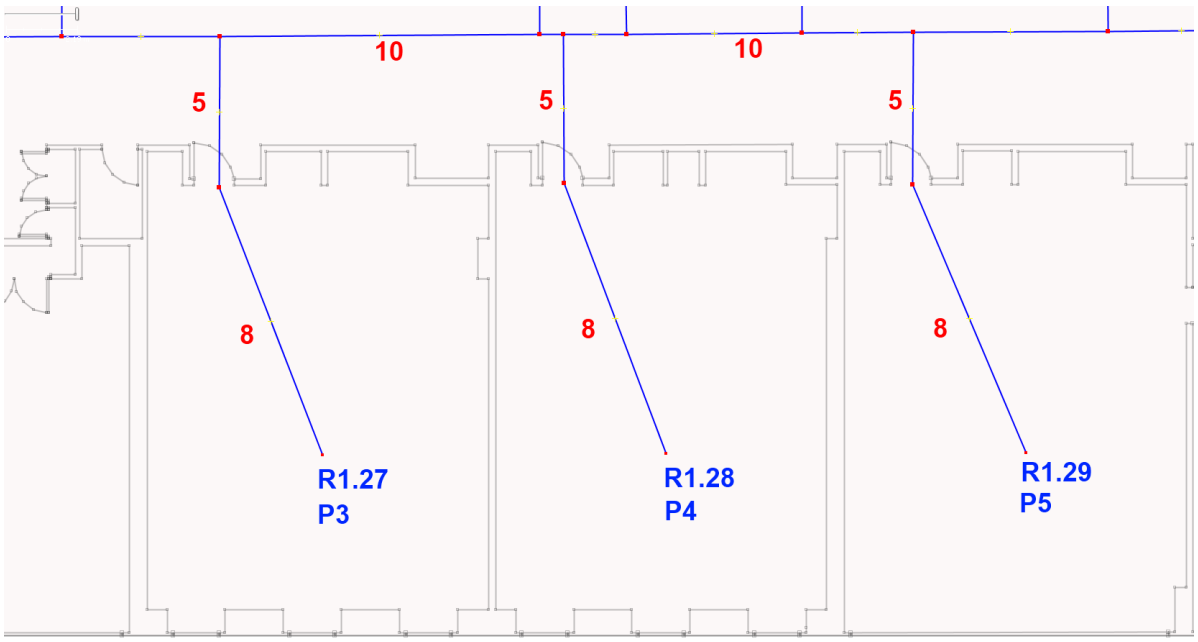
Za določitev lokacij prostorov in poti med njimi smo ustvarili utežen graf, kjer vsako vozlišče predstavlja določeno točko na zemljevidu. Graf smo izdelali v programu JOSM (Slika 4.9), tako da je vsak OSM element »node« predstavljal vozlišče. Poleg vhodov v prostore smo definirali lokacije oznak prostorov in vmesne točke, ki sestavljajo poti med prostori v stavbi. Povezavam med vozlišči smo morali nastaviti uteži zaradi iskanja in prikaza najkrajše poti med dvema lokacijama. Uteži povezav grafa smo določili z evklidsko razdaljo (4.1) med vozlišči. Koordinate točke smo pridobili iz zapisa OSM, saj je imelo vsako vozlišče, ki smo ga ustvarili v programu JOSM, nastavljene svoje koordinate v Zemljinem koordinatnem sistemu (standard WGS 84).

$$d(a,b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2} \tag{4.1}$$

Poleg koordinat smo morali nastaviti dodatne lastnosti vozliščem, ki določajo dodatne lastnosti. Vsakemu vozlišču smo dodali oznako, ki je določila nadstropje, v katerem se nahaja. Vozliščem, ki so predstavljala vhod v določen prostor, smo dodali enolično identifikacijsko šifro. Šifre prostorov smo pridobili iz arhitekturnih načrtov, kjer so bili vsi prostori primerno označeni. Ker smo graf izdelali za vsako nadstropje posebej, smo morali označiti še točke, ki povezujejo nadstropja med seboj. V primeru stavbe fakultete so povezave med nadstropji predstavljala tri stopnišča in dvigalo. Za določitev vozlišč smo uporabili naslednje OSM oznake:

- *vID=[Rx.xx]:* Identifikator prostora, ki ga predstavlja vozlišče. V primeru vmesnih vozlišč, ki ne določajo prostora, smo identifikator nastavili na generično vrednost.
- *level=[1,2,3]:* Označuje nadstropje, v katerem se nahaja vozlišče.
- *levelChange=yes:* Oznaka, ki določa vozlišče, ki predstavlja prehod med nadstropji. To oznako smo potrebovali za povezovanje grafov poti v posameznih nadstropjih.
- *roomLabel=yes:* Vozlišča s to oznako določajo lokacijo prikaza imen prostorov na zemljevidu. Ta vozlišča smo za boljšo vidljivost postavili v sredino prostorov.

Ustvarjene grafe smo shranili v posamezne datoteke OSM in jih dodali v projekt. V aplikaciji smo podatke grafov prebrali z uporabo orodij za branje XML datotek, ki so že vključena v zbirko programskih knjižnic Android SDK.



Slika 4.9 Prikaz grafa povezav med prostori. Vrednosti uteži vmesnih povezav so zaokrožene zaradi večje preglednosti.

Poglavje 5 Postopek razvoja aplikacije

5.1 Struktura aplikacije

Pri razvoju aplikacij moramo upoštevati vnaprej določeno drevesno strukturo projekta. Pri tem je programska logika ločena od predstavitvene plasti in zunanjih sredstev, ki jih potrebuje aplikacija za delovanje (slike, ikone zvočne datoteke). Tako je vsaka aplikacija že v osnovi zgrajena modularno, zato se lahko prilagaja različnim vrstam naprav in omogoča enostavno prevajanje v druge jezike. Pri vzpostavitvi pravilne strukture projekta nam pomaga razvojno okolje Android Studio, ki ob ustvarjanju projekta ustvari tudi primerno drevesno strukturo. Projekt naše aplikacije ima sledečo strukturo:

```
friMap (ime projekta)
       |- libs (mapa v kateri se nahajajo uporabljene programske knjižnice)
       |- src/main
              |- assets (sredstva, ki jih potrebuje aplikacija)
              |- java (v tej mapi se nahaja vsa programska koda aplikacije)
              |- res (tudi tukaj nahajajo sredstva za uporabo v aplikaciji vendar se
              datoteke  prevedejo v drugačno obliko in indeksirajo za lažji dostop v
              programski kodi)
              |- AndroidManifest.xml (nastavitvena datoteka)
```

Programska sredstva, ki jih potrebuje aplikacija, lahko shranimo v mapo »assets« ali »res«. V prvi so datoteke shranjene kot v običajnem datotečnem sistemu, v drugi pa se datoteke samodejno indeksirajo za lažji dostop v programski kodi. Večino sredstev smo zato shranili v mapo »res«, v mapo »assets« smo shranili le nastavitveno datoteko rendertheme.xml, saj je tako zahtevala knjižnica Mapsforge. Mapa s programskimi sredstvi »res« je še naprej ločena na več podmap. Naslovi le teh so že vnaprej določeni in se razlikujejo glede na vrsto datotek, ki jih vsebujejo. Nekatere mape so določene glede na lastnosti naprave, na kateri teče aplikacija (velikost zaslona, orientacija zaslona, ločljivost). Druge so določene glede na trenutno stanje telefona (orientacija zaslona, nastavljen jezik). Na ta način v programski kodi ni potrebno nalagati različnih sredstev glede na lastnosti naprave, ampak to sistem naredi samodejno. Upoštevati moramo samo pravilen zapis map. Zaradi velikega števila možnih zapisov map bomo našteli le tiste, ki smo jih potrebovali v naši aplikaciji:

- drawable: Mapa, v kateri se nahajajo slikovne datoteke.
- layout: Vsebuje datoteke, ki definirajo postavitev uporabniškega vmesnika

- menu: Vsebuje datoteke za nastavitev uporabniških menijev, ki se pojavijo v aplikaciji
- raw: Ostale datoteke, ki smo jih uvozili v aplikacijo. Tukaj smo shranili datoteke z načrti nadstropij stavbe in podatki o zasedenosti prostorov.
- values: Datoteke, kjer so nastavljene vrednosti enostavnih tipov, kot so nizi, seznami, definicije barv in ostalo.
- xml: Druge xml datoteke. V primeru naše aplikacije se je tukaj nahajala datoteka z nastavitvami iskalnika prostorov.

Ustvarimo lahko tudi dodatne mape s priponami, ki označujejo lastnosti naprave. V primeru, če želimo uporabiti slike z višjo ločljivosti za naprave z zasloni z visoko gostoto pik, moramo ustvariti dodatno datoteko drawable in ji določiti primerno pripono. Podobno lahko storimo pri vseh ostalih mapah.

Datoteke s programsko kodo aplikacije se v celoti nahajajo v mapi main/java. Razvijalec ni omejen pri poimenovanju datotek in razdelitvijo datotek v različne pakete. Pri razvijanju aplikacije smo datoteke ločili v pakete, tako da je bila koda urejena glede na njen namen.

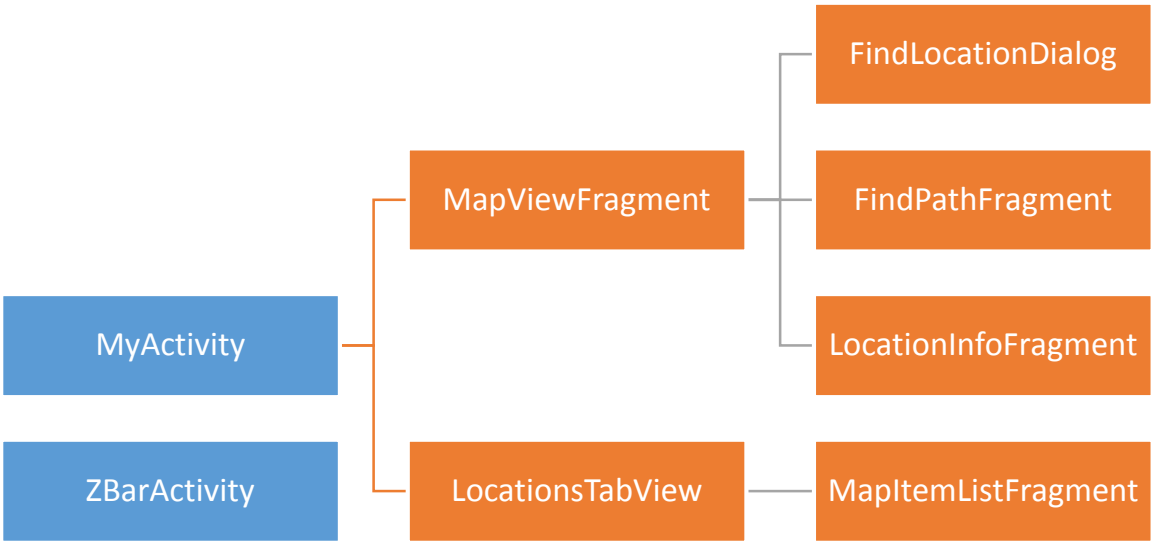
```
java/si.uni_lj.fri.friMap (ime osnovnega paketa)
       |- adapters (adapterji služijo pretvarjanju podatkov v obliko, ki jo lahko
       prikažemo v uporabniškem vmesniku)
       |- data (koda za branje in reprezentacijo podatkov o prostorih)
       |- fragments (programska koda fragmentov)
       |- graph (programska koda za branje in shranjevanje podatkov o poteh v
       prostoru)
       |- interfaces (vmesniki služijo medsebojni komunikaciji med fragmenti in s
       pripadajočo aktivnostjo)
       |- MainActivity.java (Aktivnost, ki jo uporabljamo v aplikaciji)
       |- MyApplication.java (Vhodna točka aplikacije. Večinoma te kode ni potrebno
       dodajati. V primeru naše aplikacije smo morali uporabiti to datoteko za
       nastavitev knjižnice ACRA za beleženje programskih napak)
       |- Utils.java (Preostale splošno namenske metode, ki jih kličemo v
       aplikaciji)
```

Pri snovanju aplikacije smo upoštevali modularen pristop z uporabo fragmentov, ki so samostojne enote uporabniškega vmesnika. Glavni fragment, kjer se nahaja večino programske logike, je `MapViewFragment`. V njem se nahaja koda za prikaz tlorisa in ostale funkcije povezane z njim. V njegov uporabniški vmesnik so dodani trije drugi fragmenti, ki jih potrebujemo za različne funkcije. `LocationInfoFragment` uporabljamo za izpis informacij o trenutno izbrani lokaciji oziroma poti, če je trenutno določena. `FindLocationDialog` je izbirno okno, kjer uporabnik izbere način, s katerim bo našel lokacijo. V fragmentu `FindPathFragment` uporabnik določi začetno in končno lokacijo. S klikom na eno od lokacij se prikaže `FindLocationDialog`.

`LocationsTabView` je fragment, ki vsebuje sezname lokacij. Razdeljen je v več fragmentov tipa `MapItemListFragment`, ki so razporejeni v zavihke. Vsak fragment `MapItemListFragment` vsebuje seznam prostorov določene kategorije.

`ZBarActivity` je ločena aktivnost, ki se zažene, ko uporabnik želi odčitati QR zapis za prepoznavo lokacije.

Slika 5.1 prikazuje strukturo aplikacije. Z modro sta obarvani glavni aktivnosti, z rdečo pa so označeni vsi fragmenti, ki jih aktivnost vsebuje.



Slika 5.1 Prikaz aktivnosti in fragmentov, ki sestavljajo uporabniški vmesnik aplikacije.

5.2 Prikaz tlorisa stavbe

Prikaz tlorisa zgradbe je eden izmed temeljnih funkcij, ki jih omogoča naša aplikacija. Za način prikaza tlorisa smo imeli možnost izbire več različnih tehnologij. Odločili smo se za odprtokodno knjižnico Mapsforge, ki omogoča prikaz zemljevidov v podobni obliki, kot je to v najbolj priljubljeni aplikaciji Google Maps. Med funkcije, ki jih omogoča knjižnica, spadajo: povečevanje in premikanje zemljevida s pomočjo dotika zaslona, nalaganje zemljevidov neposredno iz naprave in določanje načina prikaza kartografskih elementov z uporabo nastavitvene datoteke `rendertheme.xml`.

5.2.1 Nastavitev prikaza elementov tlorisa

Datoteka rendertheme.xml vsebuje ukaze, ki nam omogočajo nastavitev končnega izgleda zemljevida neposredno iz aplikacije. Zaradi tega je lahko prikaz zemljevida prirejen potrebam vsake aplikacije posebej. Na voljo imamo različna pravila, s pomočjo katerih lahko določimo

element zemljevida in v kakšni obliki se bo izbrani element prikazal. V tabeli 1 so našteti vsi atributi, s katerimi izberemo določen kartografski element.

Atribut	Možne vrednosti	Opis	Potreben element?
E	Node,way,any	S tem atributom določimo ustrezne tipe elementov.	Da
K	Besedilo	Ključ oznake, ki se uporablja kot filter. Z zvezdico označimo vse možne oznake. S pokončno črto lahko ločimo več oznak.	Da
V	Besedilo	Vrednost oznake.	Da
Closed	Yes,no,any	Določimo, katere poti so primerne. Zaprte ali odprte.	Ne
Zoom-min	0-255	Najmanjša vrednost povečave zemljevida, pri katerem se prikaže element.	Ne
Zoom-max	0-255	Najvišja vrednost povečave.	Ne

Tabela 1 Atributi za filtriranje elementov in opis njihovih vrednosti

Obstaja več ukazov, ki določajo, kako se bo element prikazal na zemljevidu. Vsak element lahko vsebuje enega ali več naslednjih ukazov:

- area (površina)
- line (črta)
- lineSymbol (črta, ki uporabi png sliko za prikaz)
- symbol (prikaže ikono)
- caption (besedilo)
- circle (krog)
- pathText (besedilo, ki sledi črti)

Na sliki 5.2 je prikazana vsebina datoteke rendertheme.xml z ukazi, ki smo jih uporabili v naši aplikaciji. Iz zapisa lahko razberemo, kako smo filtrirali vsak posamezni element zemljevida

(zidovi, okna, vrata) in jim določili barvo polnila, barvo obrobe ter debelino obrobe. Ukazi, ki smo jih uporabili so:

- Area fill: barva polnila izbrane površine. Ukaz lahko uporabimo le pri zaprtih poteh.
- Stroke: barva robov površine
- Line stroke: barva drugih odprtih poti, ki ne predstavljajo površin
- Stroke width: debelina robov in poti

```
<?xml version="1.0" encoding="UTF-8"?>
<rendertheme xmlns=http://mapsforge.org/renderTheme
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://mapsforge.org/renderTheme ../renderTheme.xsd"
version="4" map-background="#f8f8f8" >

 <rule e="way" k="*" v="*">

  <rule e="way" k="building" v="*">
    <area fill="#f3d6b6" stroke="#6a5a8e" stroke-width="0.02" />
  </rule>

  <rule e="way" k="indoor" v="*">
   <rule e="way" k="indoor" v="wall">
     <area fill="#7e7e7e" stroke="#797978" stroke-width="0.01" />
   </rule>

   <rule e="way" k="indoor" v="window">
     <area fill="#05449B" stroke="#6a5a8e" stroke-width="0.01" />
     <line stroke="#05449B" stroke-width="0.01" stroke-linecap="butt" />
   </rule>

   <rule e="way" k="indoor" v="door">
     <line stroke="#612500" stroke-width="0.03" stroke-linecap="butt" />
   </rule>
  </rule>

  <rule e="way" k="stairs" v="yes">
    <line stroke="#235F06" stroke-width="0.02" stroke-linecap="butt" />
  </rule>

  <rule e="way" k="highway" v="elevator">
    <line stroke="#235F06" stroke-width="0.02" stroke-linecap="butt" />
  </rule>

 </rule>
</rendertheme>
```

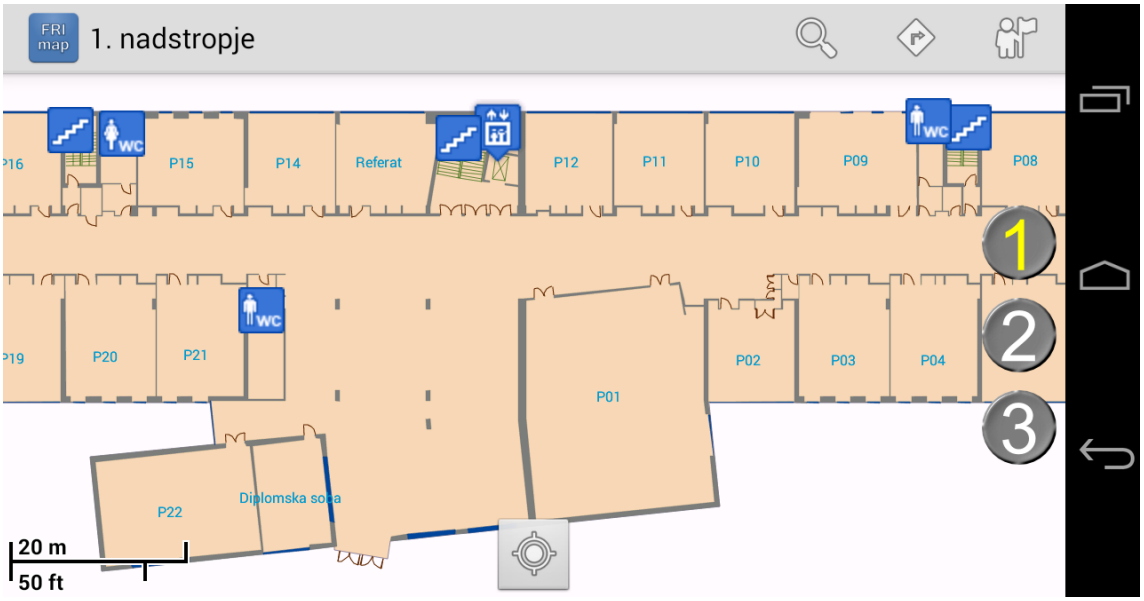
Slika 5.2 Vsebina nastavitvene datoteke rendertheme.xml

5.2.2 Nastavitev tlorisa v programski kodi

Programsko kodo za prikaz tlorisa smo zapisali v fragment `MapViewFragment`. V tem fragmentu smo zapisali večino programske logike za delovanje osnovnih funkcij aplikacije, kot so prikaz tlorisa z izbiro nadstropja, prikaz imen prostorov in ikon, ki predstavljajo pomembne

lokacije in prikaz poti na tlorisu. Najprej smo morali pridobiti referenco do objekta `MapView`, ki ponuja vse metode za interakcijo s tlorisom. Ob prvem zagonu aplikacije, oziroma ko aktivnost preide v aktivno stanje, je bilo potrebno pravilno nastaviti tloris. Nastaviti je bilo potrebno začetno pozicijo in dovolj visoko povečavo, da je tloris stavbe dovolj dobro viden. Prav tako smo omejili pomikanje, tako da uporabnik ne more premakniti prikaza tlorisa izven območja stavbe fakultete. Pri nastavljanju prikaza tlorisa je bilo potrebno naložiti ustrezne datoteke, v katerih so bili shranjeni načrti. Naložili smo tri datoteke, ki predstavljajo posamezna nadstropja, vsako v svojo plast, pri čemer se prikazuje samo tisto nadstropje, ki je tisti čas izbrano. Vsaka plast, v kateri je naloženo določeno nadstropje, ima definiran tudi predpomnilnik (cache), ki omogoča hitrejši izris. Vse nastavitve prikaza tlorisa smo zapisali v metodi `initialize()`, ki se požene ob zagonu, oziroma ko fragment `MapViewFragment` pride v aktivno stanje. Za izbiro nadstropja, ki se prikazuje v nekem trenutku, smo v vmesnik dodali tri gumbe. Z zabeleženim dotikom na enem od teh gumbov se nato kliče ustrezna metoda za spremembo nadstropja, ki se prikazuje. Na sliki 5.3 je končni prikaz tlorisa med delovanjem aplikacije.

Pri prikazu tlorisa je bilo potrebno paziti na porabo pomnilnika aplikacije. Ker ima vsaka plast svoj predpomnilnik, ki zavzema nekaj prostora, je bilo potrebno med menjavo nadstropja počistiti prej uporabljeni predpomnilnik. Predpomnilnik je bilo potrebno sprazniti oziroma uničiti tudi ob ustavitvi in odstranitvi fragmenta iz uporabniškega vmesnika (v metodah `onPause()` in `onDestroyView()`).

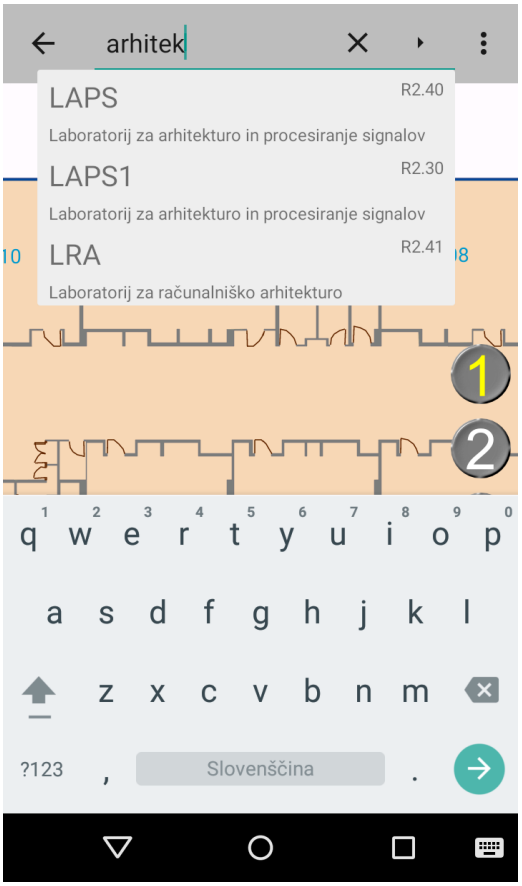


Slika 5.3 Prikaz tlorisa fakultete

5.3 Seznam in iskalnik prostorov

Za učinkovito uporabo aplikacije potrebujejo uporabniki hiter pregled nad vsemi prostori, ki se nahajajo v stavbi fakultete. V ta namen smo uporabili dva gradnika. Prvi je bil seznam prostorov, ki je bil razdeljen na zavihke glede na kategorijo prostora in razvrščen po abecednem redu. Drugi gradnik je bil iskalno polje, ki je omogočal hitro iskanje določenega prostora po imenu (Slika 5.4). Iskalnemu polju smo dodali funkcijo prikazovanja seznama najbolj primernih rezultatov glede na trenuten niz, ki ga je uporabnik vpisal v iskalno polje (autocomplete).

Za iskanje in prikaz podatkov o prostorih smo morali najprej prebrati podatke iz datotek CSV in jih zapisati v obliko, ki jo lahko uporabimo v aplikaciji. Za predstavitev podatkov v aplikaciji smo ustvarili nov abstraktni razred `MapItem`, ki je vseboval vse skupne lastnosti prostorov. Vsaka kategorija prostora je imela ustvarjen svoj razred, ki je razširjal osnovni razred `MapItem` in je imela definirane dodatne lastnosti in metode, ki so značilne za to kategorijo prostora. Podatki se iz datotek CSV preberejo ob vsakem zagonu aplikacije in se shranijo v seznam, ki vsebuje vse elemente tipa `MapItem`.



Slika 5.4 Delovanje iskalnika prostorov

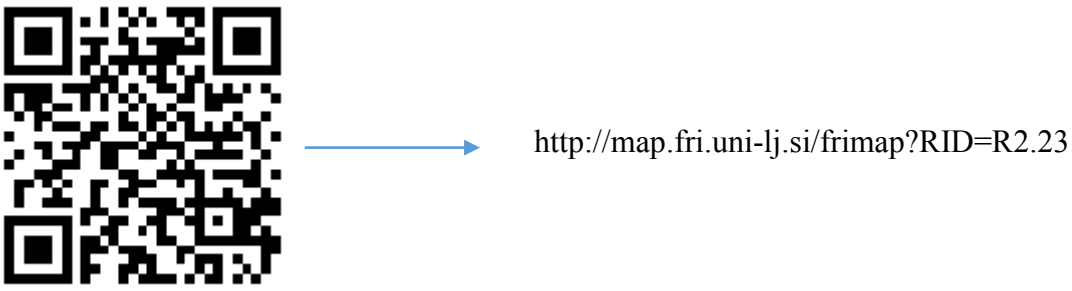
Za prikaz seznama elementov določenega tipa se v sistemu Android uporablja razred `ListAdapter`. V primeru, če želimo ustvariti svoj način predstavitve, moramo ustvariti nov razred, ki razširja osnovni razred `ListAdapter`. Od vseh metod, ki jih moramo razširiti, je najpomembnejša `getView()`, v kateri določimo podatke, ki se izpišejo v vrstici seznama.

Za prikaz zavihkov različnih kategorij prostorov smo uporabili zunanjo knjižnico `PagerSlidingTabStrip`. Knjižnica deluje v kombinaciji z osnovnim gradnikom `ViewPager`, ki omogoča premikanje med več stranmi uporabniškega vmesnika, z drsanjem levo ali desno po zaslonu. `PagerSlidingTabStrip` temu gradniku doda še zavihke z naslovi, ki omogočajo lažje iskanje po različnih straneh. V primeru, ko uporabnik pritisne na vrstico v seznamu, se sproži dogodek, ki informacijo o izbrani lokaciji posreduje fragmentu, ki vsebuje prikaz tlorisa. Ta nastavi izbrano lokacijo v središče zaslona in izpiše informacije o njej.

Za iskanje prostora in oseb po imenu smo uporabili element `SearchView`, ki je na voljo v zbirki Android SDK. `SearchView` ustvari iskalno polje v zgornji vrstici aplikacije, kjer so med drugim gumbi za navigacijo po aplikaciji, ikona ter naslov aplikacije. Pri iskanju se pojavi seznam rezultatov, ki so primerni glede na nedokončan niz. Uporabnik lahko hitreje dokonča iskalno poizvedbo, tako da klikne na primeren rezultat v seznamu. `SearchView` smo v naši aplikaciji uporabili v glavnem fragmentu `MapViewFragment`. V metodi `onCreateOptionsMenu()` smo nastavili element in poslušalce dogodkov. V primeru spremembe niza, se kliče metoda za iskanje prostorov in oseb z enakim podnizom v imenu. Rezultat se vrne kot seznam primernih prostorov, ki se prikaže pod iskalnim poljem.

5.4 Prepoznavanje lokacije s pomočjo kod QR

Preden se lahko uporabnik uspešno orientira na tlorisu, ki ga ponuja aplikacija, potrebuje namig o tem, kje se trenutno nahaja v prostoru. Velikokrat je težko primerjati okolico, v kateri se uporabnik nahaja z reprezentacijo v aplikaciji. Še posebno se je težko orientirati v velikih stavbah s kompleksno razporeditvijo prostorov. V naši aplikaciji smo uporabili enostaven sistem, kjer uporabnik z vgrajeno kamero prebere zapis zakodiran v kodi QR (*Quick Response*), ki so razporejeni na vidnih in pogosto obiskanih mestih v stavbi. QR kodiran zapis nam omogoča, da se tudi ob pojavu šuma v ozadju z veliko mero natančnosti pridobi podatke, ki so bili kodirani. Večina mobilnih naprav ima vgrajeno kamero, tako da je lahko ta funkcija aplikacije dostopna večini uporabnikom.



Slika 5.5 Primer prepoznave lokacije iz kode QR. Prebrani zapis ima funkcijo opisano v nadaljnjem besedilu.

Zapis, ki smo ga zakodirali v kodo QR, ima dvojno funkcijo. V primeru, če uporabnik še nima naložene aplikacije »Fri map«, ga zapis napoti na spletni naslov, kjer si lahko aplikacijo naloži. V nasprotnem primeru je bil zapis prebran s strani aplikacije in se iz njega izlušči podatek o lokaciji. Slika 5.5 prikazuje primer prepoznavanja lokacije iz zapisa, ki ga pridobimo iz kode QR. Lokacijo pridobimo tako, da najprej iz zapisa izluščimo šifro prostora, nato pa lokacijo prikažemo na tlorisu.

Za odčitavanje kode QR smo uporabili knjižnico `BarcodeScanner` in njen modul `ZBar`. Knjižnica ponuja enostaven uporabniški vmesnik s prikazano sliko, ki jo v tistem trenutku zajema kamera telefona. Večji del zaslona v sredini prekriva pravokotnik, ki opredeljuje območje, kjer se samodejno prepozna koda (Slika 5.6). `ZBar` lahko prepozna večje število standardnih kod, med katerimi je tudi QR. V naši aplikaciji smo knjižnico uporabili tako, da smo ustvarili novo aktivnost `ZBarActivity`, ki ima prej omenjeni uporabniški vmesnik. Sistem Android je zgrajen tako, da lahko aktivnost zaženemo tudi neposredno iz druge aktivnosti, nato pa čakamo na rezultat. Tako smo v naši aplikaciji v glavni aktivnosti `MyActivity` ob dogodku, ko hoče uporabnik odčitati kodo, zagnali aktivnost `ZBarActivity`. Ko se koda uspešno prebere ali uporabnik prekliče branje, se prvi aktivnosti pošlje koda, ki označuje, kateri dogodek se je zgodil. Za pošiljanje podatkov med aktivnostmi smo uporabili objekt tipa `Intent`. Ob uspešnem odčitku smo v ta objekt zapisali dekodiran zapis in ga poslali nazaj v glavno aktivnost, ki ga pridobi v metodi `onActivityResult()`.

Slika 5.6 Prikaz branja kode QR v aplikaciji

5.5 Pomoč pri navigaciji v notranjosti stavbe

Zadnji pripomoček, ki uporabniku olajša iskanje določenega prostora v stavbi, je prikaz poti od začetne izbrane lokacije do cilja. Začetno in končno lokacijo lahko pridobimo z enim od prej opisanih načinov: iskalnik, izbira iz seznama ali branje kode QR. Za določanje obeh lokacij smo v aplikacijo dodali izbirno okno, kjer določimo lokaciji glede na prej omenjene možnosti. Izbirno okno je del uporabniškega vmesnika in je ob zagonu aplikacije skrito, ob kliku na gumb za določanje poti pa se prikaže na sredini okna. Programska logika izbirnega okna je shranjena v fragmentu `FindPathFragment`.

Prostori v stavbi fakultete, za katero je bila ustvarjena aplikacija, imajo sorazmerno enostavno postavitev. Med dvema lokacijama redko izbiramo med več različnimi poti. Vendar je za ljudi, ki še ne poznajo postavitve prostorov v stavbi, lažje če dobijo dodatno informacijo o vmesni poti. Za določanje poti smo uporabili graf poti, ki smo ga prej sestavili z orodjem JOSM in shranili v datoteko OSM. Postopek izdelave smo opisali v poglavju 4.4. Za branje podatkov iz datoteke smo uporabili orodja za branje datotek XML, ki so sestavni del skupka knjižnic Android SDK. Pri branju smo morali razlikovati med dvema osnovnima elementoma, ki sta bila

vsebovana v datoteki. To sta pot in vozlišče. Pri branju vozlišča smo ustvarili nov objekt `Vertex` (Tabela 2), ki je vseboval vse lastnosti vozlišča in ga dodali v seznam vseh vozlišč. Poleg tega smo dodali še nekaj dodatnih lastnosti, ki smo jih potrebovali kasneje pri določanju najkrajše poti (vrednost najkrajše razdalje in kazalec na prejšnje vozlišče na poti). Pri branju poti smo vsako vozlišče dodali na seznam sosedov, ki je vsebovan v nasprotnem vozlišču.

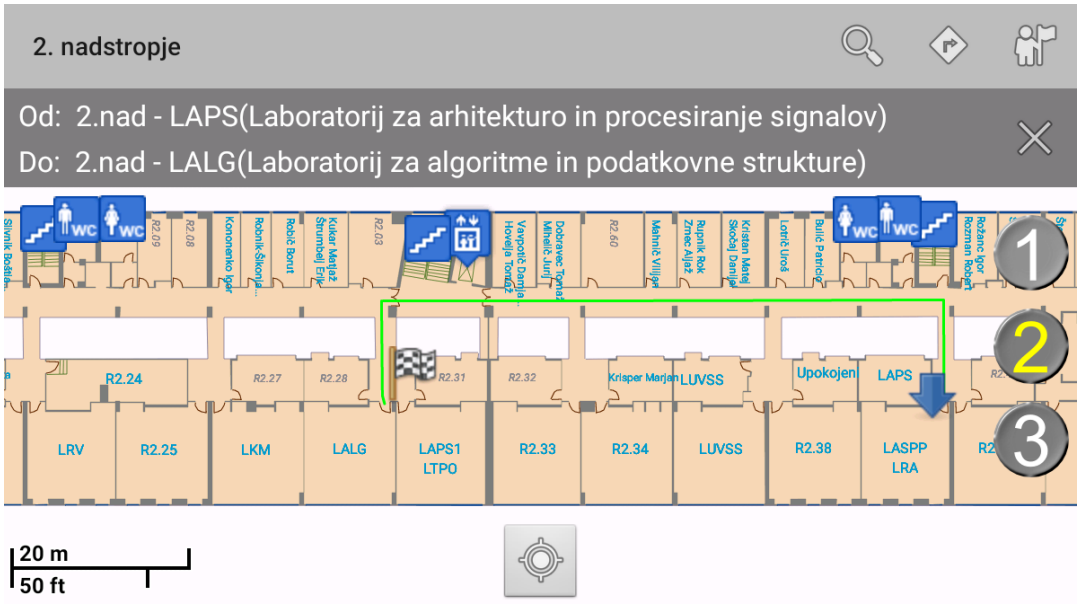
Vertex		
String	rId	Identifikator prostora, ki ga predstavlja vozlišče. V primeru drugih vozlišč je vrednost nastavljena na generično vrednost »v1.0«.
LatLong	Position	Koordinate točke v formatu WGS 84
Int	Level	Nadstropje, v katerem se vozlišče nahaja
Boolean	lvlChange	Označuje ali vozlišče predstavlja povezavo med nadstropji (stopnice ali dvigalo)
Boolean	isRoomLabel	Vozlišča s to oznako določajo točko, kjer se prikaže oznaka prostora na tlorisu
Double	minDist	Vrednost, ki jo potrebujemo za izračun najkrajše poti z algoritmom Dijkstra. Označuje najmanjšo razdaljo do vozlišča.
Vertex	previous	Kazalec do predhodnega vozlišča na izračunani najkrajši poti.
ArrayList<Vertex>	neighbors	Seznam vseh sosednjih vozlišč v grafu

Tabela 2 Podatki, ki jih vsebuje vsako vozlišče

Najkrajšo pot smo pridobili s pomočjo algoritma Dijkstra [13] , ki poišče najkrajšo pot med izbranim vozliščem in vsemi ostalimi vozlišči v grafu. Pogoj za delovanje algoritma je, da povezave med vozlišči nimajo nastavljenih negativnih uteži. V primeru grafa, ki smo ga uporabili v aplikaciji, smo za uteži nastavili evklidsko razdaljo med dvema točkama na zemljevidu. Algoritem Dijkstra smo lahko uporabili, saj je vrednost razdalje med dvema različnima točkama lahko le pozitivna. Algoritem deluje po naslednjem postopku:

1. Vsem vozliščem v grafu, razen začetnemu, nastavimo najkrajšo razdaljo na najvišjo možno vrednost (v Javi določena kot `Double.POSITIVE_INFINITY`) in jih dodamo v množico prostih vozlišč.
2. Iz prostih vozlišč izberi tistega z najmanjšo razdaljo in ga odstrani iz množice prostih vozlišč.
3. Izračunaj razdaljo do vsakega od sosedov. Razdalja je vsota uteži povezave med dvema vozliščema in že izračunane razdalje do prvega vozlišča. V primeru, če je razdalja manjša kot obstoječa razdalja do sosednjega vozlišča, zamenjaj razdaljo z izračunano in dodaj povezavo do prvega vozlišča.
4. Če množica prostih vozlišč ni prazna se vrni na korak 2.
5. Najdene so bile najkrajše poti med začetnim vozliščem in vsemi ostalimi vozlišči. Za sestavo najkrajše poti do cilja, izberi ciljno vozlišče in sledi povezavam nazaj do začetnega.

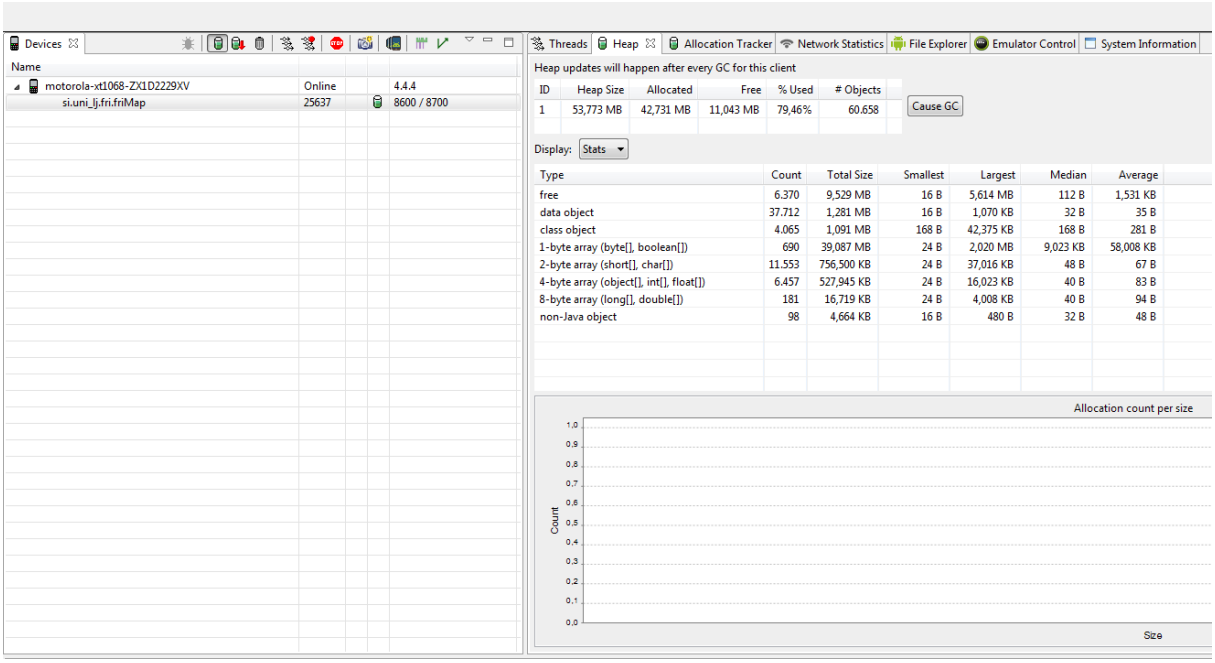
Rezultat algoritma je seznam vseh vozlišč, ki so sestavni del celotne najkrajše poti med dvema lokacijama. Pot smo na tlorisu prikazali s pomočjo knjižnice Mapsforge, ki ima možnost risanja različnih elementov neposredno na prikaz zemljevida. Za izris poti smo uporabili grafični element `PathPolyLine`, ki za prikaz potrebuje seznam lokacij vseh točk, ki sestavljajo pot. Prikaz še ni bil dovolj uporaben, saj iz njega ni bilo razvidno, kje se pot začne in konča. Zaradi tega smo dodali še ikone, ki označujejo pričetek in konec poti. Poleg tega smo dodali še ikono, ki označuje prehod med različnimi nadstropij. S klikom na to ikono se zamenja nadstropje z naslednjim, na katerem se nadaljuje pot. Na sliki 5.7 lahko vidimo, kako izgleda prikaz poti med delovanjem aplikacije.



Slika 5.7 Prikaz poti od začetne lokacije do izbranega cilja

5.6 Testiranje aplikacije

Preden smo aplikacijo objavili in ponudili končnim uporabnikom, smo morali poskrbeti za odpravo morebitnih programskih napak, ki se lahko pojavijo med uporabo aplikacije. Najtežje odpravljive napake so tiste, ki se pojavijo le ob nekaterih pogojih uporabe in jih je težko predvideti. V ta namen je v programski zbirki Android SDK na voljo nekaj orodij, ki nam olajšajo delo testiranja. Android Emulator je orodje, ki nam omogoča posnemanje katerekoli mobilne naprave na našem računalniku. Tako nam ni potrebno testirati delovanja aplikacije neposredno na vseh možnih fizičnih napravah, ampak samo zaženemo virtualno kopijo določene naprave. Edina slabost testiranja na tak način je slaba odzivnost sistema na virtualni napravi. To lahko zmoti zlasti v multimedijskih aplikacijah, ki uporabljajo veliko animacij in morajo biti zelo odzivne za normalno uporabo. Vendar je za testiranje osnovne funkcionalnosti razvite aplikacije na večjem številu različnih naprav ta način zadostovalo.

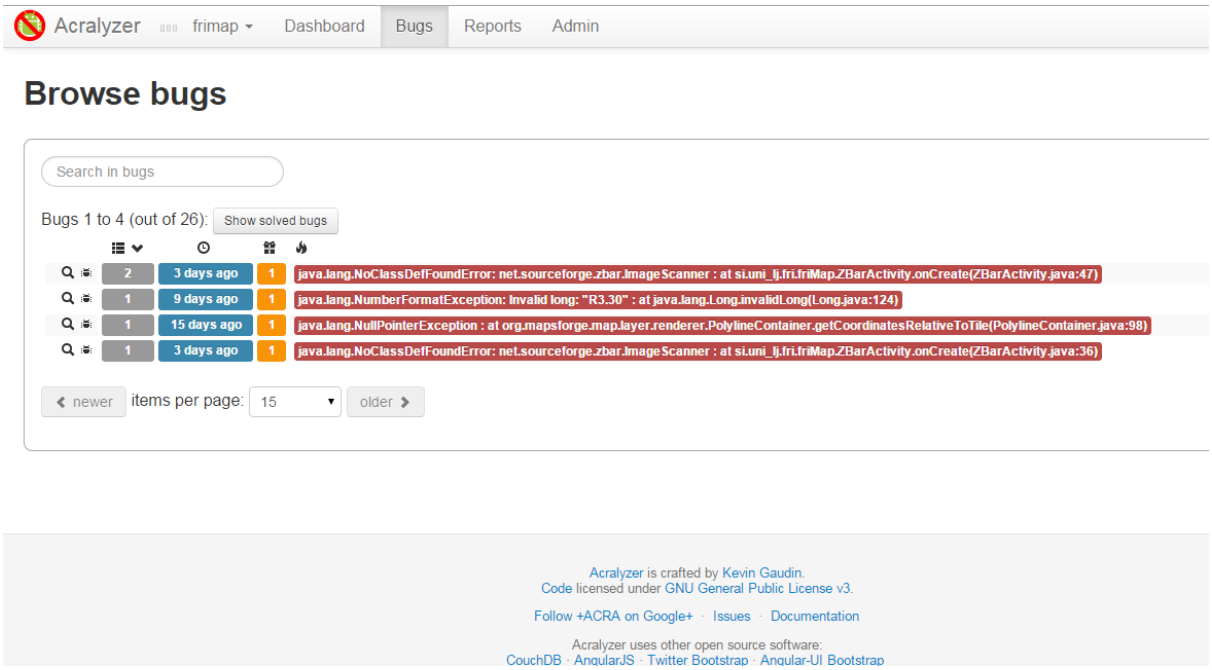


Slika 5.8 S pomočjo orodja DDMS lahko spremljamo porabo pomnilnika aplikacije in njenih sestavnih delov

Drugo pomembno orodje, ki smo ga uporabili pri odkrivanju in odpravljanju napak, je DDMS (*Dalvik Debug Monitor Server*). Na sliki 5.8 je prikazan uporabniški vmesnik orodja. Z orodjem lahko spremljamo in beležimo porabo različnih sistemskih sredstev s strani aplikacije. Pri testiranju smo ga uporabili za beleženje porabe pomnilnika. V času razvoja aplikacije smo zaznali napako, ko je aplikacija v nedogled povečevala porabo pomnilnika, dokler se zaradi prekoračitve omejitve ni ustavila. Z vpogledom v podatke, ki smo jih beležili z orodjem, smo

ugotovili, da je bila napaka v tem, da nismo pravilno počistili predpomnilnika (cache), ki ga je uporabljal prikaz tlorisa stavbe. Napako smo odpravili s klicem ustreznih metod za čiščenje predpomnilnika, ki so bile na voljo v knjižnici Mapsforge.

Aplikacija je bila proti koncu razvoja testirana na večjem številu naprav pri različnih uporabnikih. Za lažje beleženje napak smo potrebovali orodje, ki beleži napake in nam jih sporoči. Uporabili smo knjižnico ACRA (Application crash report for Android), ki ob vsaki napaki, zaradi katere se je prekinilo delovanje aplikacije, shrani poročilo napake in ga pošlje na strežnik, ki je bil že vnaprej nastavljen. V poročilu so shranjene vse informacije, ki jih potrebujemo za uspešno odkrivanje napak in njihovo odpravo. Za beleženje poročil obstaja nekaj različnih orodij. Uporabili smo spletno aplikacijo Acralyzer, ki za delovanje potrebuje samo povezavo s podatkovno bazo CouchDB. Za gostovanje aplikacije smo uporabili ponudnika Iriscouch, ki za manjšo mesečno porabo ne zaračunava stroškov. Z uporabo tega orodja smo lahko hitro poiskali programske napake, ki so se pojavile med delovanjem aplikacije pri testnih uporabnikih (Slika 5.9).



Slika 5.9 Spletna aplikacija Acralyzer, ki beleži poročila programskih napak aplikacije

5.7 Objava in posodabljanje aplikacije

V zadnjem koraku razvoja smo morali aplikacijo ponuditi končnim uporabnikom. Vsako aplikacijo, ki jo razvijemo za platformo Android, lahko ponudimo v obliki arhivske instalacijske datoteke APK, ki jo ustvarimo z orodjem Android Studio. Uporabniki morajo v primeru posodobitve ročno prenesti in naložiti novejšo različico aplikacije[2]. Poleg tega je potrebno v nastavitvah sistema označiti opcijo, ki omogoča nalaganje aplikacij iz tako imenovanih nepreverjenih virov.

5.7.1 Platforma Google Play

Bolj priročen kanal za distribucijo je platforma Google Play, preko katere lahko uporabniki najdejo vse potrebne informacije o aplikaciji, preden se odločijo za prenos oziroma nakup v primeru, če je aplikacija plačljiva. Aplikacije prenesene preko te platforme se posodobijo samodejno, zato uporabnikom ni potrebno skrbeti glede tega. Med razvijanjem aplikacije imamo možnost ponuditi ločeno razvojno različico aplikacije samo osebam, ki jih določimo za testiranje. Za objavo aplikacije na platformi smo uporabili fakultetni račun Google Play. Omeniti je potrebno, da moramo za objavo aplikacij na platformi plačati prijavnino v znesku 25$. Prijavnino moramo plačati samo enkrat za dostop do računa, na katerem lahko objavimo neomejeno število aplikacij. Pri nalaganju aplikacije na platformo smo morali posredovati vse dodatne podatke, s katerimi predstavimo njene funkcije končnim uporabnikom. V spletnem obrazcu smo vnesli kratek opis njenih funkcionalnosti in dodali različne slike uporabniškega vmesnika. S pomočjo teh podatkov uporabniki lažje aplikacijo poiščejo na platformi in se odločijo ali jo želijo naložiti na napravo[3].

5.7.2 Posodabljanje vsebine aplikacije

Po objavi aplikacije se lahko pojavijo dodatne napake v delovanju, ki jih med testiranjem nismo uspeli zaznati. V primeru, če smo napako zaznali in uspešno odpravili po tem, ko je bila aplikacija že objavljena na platformi, jo lahko enostavno posodobimo tako, da ponovno naložimo popravljeno različico. Vsi uporabniki, ki so že naložili aplikacijo, bodo dobili opozorilo, da je na voljo novejša in popravljena različica.

Občasno bo potrebno posodobiti tudi podatke o zasedenosti prostorov na fakulteti. Zaradi tega, ker se bo zasedenost prostorov lahko pogosto spreminjala, smo ustvarili poenostavljen sistem posodabljanja podatkov. V primeru spremembe podatkov o zasedenosti nekega prostora tako

[2] Spletni naslov kjer lahko prenesemo namestitveni paket (APK) aplikacije je: http://map.fri.uni-lj.si/frimap
[3] Google Play naslov je: https://play.google.com/store/apps/details?id=si.uni_lj.fri.friMap

ne bo potrebno neposredno spreminjati kode v aplikaciji, ampak samo datoteke, v kateri so shranjeni podatki. V poglavju 4.4 smo opisali podatkovni model za zapis podatkov o prostorih in tabele, v katerih so podatki shranjeni. Za posodobitev podatkov je potrebno spremeniti podatke v ustrezni tabeli in jo shraniti v zapisu CSV. Datoteko moramo nato uvoziti v projekt in prepisati že obstoječo datoteko, ki se nahaja v mapi `/res/raw`. Posodobljeno aplikacijo nato ponovno naložimo na platformo Google Play.

Poglavje 6 Možnosti za nadaljnje delo

Končana aplikacija je že zadoščala osnovnim ciljem, ki smo si jih zastavili v diplomski nalogi. Vendar smo med razvojem in med testiranjem opazili, da bi se v nekaterih delih aplikacijo lahko izboljšalo. Najbolj opazna pomanjkljivost razvite aplikacije je določanje trenutne lokacije naprave v stavbi. Sicer se lahko v aplikaciji lokacijo prepozna iz kod QR, vendar je to mogoče le na mestih v stavbi, kjer so postavljene kode QR. V pričujočem poglavju je omenjenih nekaj izboljšav, ki bi omogočale samodejno razpoznavo lokacije.

6.1 Določanje lokacije naprave v prostoru

Zaradi slabe kakovosti GPS signala v notranjosti stavbe ne moremo enostavno določiti trenutne lokacije uporabnika. Za rešitev tega problema je bilo predlaganih več različnih metod, ki uporabljajo signale iz okolice za določanje lokacije naprave. V [1] je opisana primerjava najbolj razširjenih metod. Metoda, ki je najpogosteje omenjena kot nadomestilo uporabi GPS sistema, je uporaba signalov brezžičnih omrežij postavljenih v notranjosti stavb [14]. Lokacijo se pridobi s pomočjo primerjanja odčitanih signalov v napravi s podatki preteklih odčitkov shranjenih v podatkovni bazi. Rezultat primerjanja je približna ocena lokacije posameznika v času odčitavanja signalov. V mobilnih napravah lahko uporabimo merjenje signalov brezžičnih omrežij in baznih postaj v kombinaciji z merilcem pospeška [2].

Velik del problema opisane metode je izgradnja računskega modela, ki natančno določa jakosti signalov na določeni lokaciji v prostoru. V [14] je opisan sistem, kjer se jakosti signalov najprej odčitajo na lokacijah, ki so enakomerno razporejene po prostoru. Te odčitke se nato shrani v bazo in kasneje uporabi v aplikaciji za določanje lokacije naprave. Slabost tega sistema je v tem, da je potrebno narediti veliko odčitkov, če želimo pokriti celoten prostor v notranjosti stavbe. Prav tako se tak sistem ne prilagaja spremembam v topologiji brezžičnega omrežja. V primeru, da se zamenja veliko brezžičnih postaj, je potrebno spremeniti bazo odčitkov in ponovno narediti odčitke v stavbi.

Drugačen sistem je opisan v [15], kjer ni potrebno neposredno zgraditi podatkovne baze odčitkov, ampak se baza sčasoma zgradi samodejno. Sistem je narejen tako, da se najprej določi lokacije v stavbi, za katere lahko z določeno mero natančnosti napovemo, kakšna bo tam

meritev signalov. V dvigalu bodo tako jakosti brezžičnih signalov slabe, medtem ko bodo imele meritve pospeška vedno enako obliko. Podobno lahko sklepamo za meritve pospeška na stopnicah in med navadno enakomerno hojo.

Zadnja metoda, ki jo bomo omenili, omogoča lociranje samo s pomočjo podatkov porabe baterije [16]. Poraba baterije naj bi bila odvisna od oddaljenosti naprave od bazne postaje in vmesnih ovir, ki zmanjšujejo jakost prejetega signala. Zaradi tega mora mobilna naprava zvišati jakost oddajanja, zato se posledično tudi zviša poraba baterije.

Pri razvoju aplikacije smo se osredotočili na robusten in stabilen prikaz samega tlorisa stavbe. Zaradi obširnosti problematike lociranja posameznika, smo v aplikacijo vgradili le sistem beleženja signalov med uporabo aplikacije. Poleg meritev jakosti brezžičnih signalov smo zajemali tudi vrednost meritev senzorja za merjenje pospeška in Zemljinega magnetnega polja. Podatke smo nato v določenih časovnih intervalih poslali na strežnik, kjer se hranijo za nadaljnjo obdelavo. Začasni strežnik smo ustvarili z orodjem Node.js [17] , ki ponuja hiter in enostaven način za vzpostavitev strežnika. Podatke smo pošiljali v zapisu JSON (Tabela 3) in ga na strežniški strani shranili v posamezne datoteke. V nadaljnjem delu bi se podatke lahko uporabilo za analizo in gradnjo podatkovnega modela, ki se bo uporabil pri določanju lokacije naprave.

Id	MAC naslov naprave za identifikacijo odčitka
Fingerprint	Vrednosti jakosti signalov brezžičnih omrežij
Orientation	Orientacija oziroma azimut naprave
Geomagnetic	Vrednost jakosti zemeljskega magnetnega polja
Linear Acceleration	Vrednost linearnega pospeška naprave
Battery	Informacije o stanju baterije (trenutna vrednost napetosti in toka)
Time	Čas odčitka

Tabela 3 Podatki meritev senzorjev in brezžičnih signalov, ki jih pošilja aplikacija

Poglavje 7 Zaključek

V diplomski nalogi smo opisali problem prikazovanja notranjosti stavb v podobni obliki, kot je to možno pri zemljevidih zunanjih površin. Tehnologija za prikaz notranjih površin je še v razvoju, zato obstaja le nekaj rešitev največjih ponudnikov aplikacij za prikaz zemljevidov. Vendar so te rešitve namenjene le prikazu javno dostopnih prostorov. Tloris prostorov preoblikujejo zaposleni v podjetjih, ki nudijo storitev, tako da lastnik stavbe ne more vplivati na obliko prikaza tlorisa. Poleg tega ima vsaka stavba svoje značilnosti, ki se razlikujejo od ostalih stavb, zato potrebuje aplikacija prirejen podatkovni model za zapis podatkov o prostorih. Po drugi strani obstoječe aplikacije za prikaz zemljevidov potrebujejo dostop do omrežja. To lahko predstavlja problem, če naprava nima ali ima omejen dostop do omrežja, kar se lahko pogosto zgodi v notranjosti stavb. Res je sicer, da aplikacije ponujajo možnost vnaprejšnjega shranjevanja dela zemljevida, vendar mora uporabnik vnaprej neposredno določiti, kateri del zemljevida naj se shrani.

V sklopu diplomske naloge smo opisali razvoj mobilne aplikacije, ki skuša rešiti omenjene probleme. Primer stavbe, za katero se prikazujejo prostori, je Fakulteta za računalništvo in informatiko v Ljubljani. Aplikacija ponuja iskanje po različnih prostorih, ki se delijo na predavalnice, laboratorije, kabinete in ostale prostore. Bistvena prednost aplikacije v primerjavi z ostalimi rešitvami je to, da je razvita za točno določen namen prikazovanja notranjosti stavbe. Tako so podatki o prostorih bolj natančni in ažurni, saj lahko lastnik aplikacije podatke sproti posodablja. Poleg tega je omogočen nadzor nad prikazom tlorisa s strani naročnika aplikacije oziroma lastika stavbe. Aplikacija ima vse podatke shranjene v napravi, zato med delovanjem ne potrebuje omrežne povezave, kar je tudi prednost v primerjavi z drugimi rešitvami.

Razvita aplikacija v končni obliki deluje zadovoljivo in uporabnikom nudi rešitev za hitro orientacijo v prostoru fakultete. Povratne informacije uporabnikov, ki so aplikacijo testirali med razvojem, so povečini pozitivne. Izboljšave, ki bi potrebovale večje spremembe v sistemu, smo omenili proti koncu diplomskega dela in predlagali možnosti za nadaljnje delo v prihodnosti. Omenili smo možnost izboljšave sistema za določanje lokacije posameznika v prostoru.

Glavni cilj diplomskega dela je bil razviti aplikacijo, ki deluje neodvisno od zunanjih ponudnikov in brez povezave v omrežje. Predlagani sistem je zadostil tem pogojem in ponuja enostavno in celovito rešitev za prikaz notranjih prostorov stavbe. Prednost sistema je tudi v

njegovi ceni, saj smo pri razvoju uporabili odprtokodno in brezplačno programsko opremo. Edini strošek pri morebitni komercialni uporabi predstavlja delo, ki bi ga morali opraviti pri pretvorbi tlorisa za posamezno stavbo in spremembi podatkov o prostorih.

Spletni naslov, kjer se nahaja namestitvena datoteka APK:
http://map.fri.uni-lj.si

Google Play naslov aplikacije:
https://play.google.com/store/apps/details?id=si.uni_lj.fri.friMap

Literatura

[1] L. Hui, H. Darabi, P. Banerjee in J. Liu, „Survey of wireless indoor positioning techniques and systems,“ *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on,* Izv. 37, št. 6, str. 1067 - 1080, 2007.

[2] M. Eladio, O. Vinyals, G. Friedland in R. Bajcsy, „Precise indoor localization using smart phones,“ v *Proceedings of the international conference on Multimedia*, Firence, Italija, 2010.

[3] OpenStreetMap, „JOSM - Download,“ [Elektronski]. Dosegljivo: http://josm.openstreetmap.de/wiki/Download. [Poskus dostopa februar 2015].

[4] Mapsforge, „Mapsforge,“ [Elektronski]. Dosegljivo: https://github.com/mapsforge/mapsforge/. [Poskus dostopa februar 2015].

[5] OpenStreetMap, „Osmosis,“ [Elektronski]. Dosegljivo: http://wiki.openstreetmap.org/wiki/Osmosis. [Poskus dostopa februar 2015].

[6] Dassault Systemes, „Download Draftsight,“ [Elektronski]. Dosegljivo: http://www.3ds.com/products-services/draftsight-cad-software/free-download/. [Poskus dostopa februar 2015].

[7] Android Open Source Project, „Dashboards,“ [Elektronski]. Dosegljivo: https://developer.android.com/about/dashboards/index.html. [Poskus dostopa januar 2015].

[8] Android Open Source Project, „Application Foundamentals,“ [Elektronski]. Dosegljivo: http://developer.android.com/guide/components/fundamentals.html. [Poskus dostopa januar 2015].

[9] W.-M. Lee, „Activites, Fragments and Intents,“ v *Beginning Android 4 Application Development*, Indianapolis, John Wiley & Sons, 2012, str. 35-41.

[10] OpenStreetMap, „OSM XML,“ [Elektronski]. Dosegljivo: http://wiki.openstreetmap.org/wiki/OSM_XML. [Poskus dostopa januar 2015].

[11] OpenStreetMap, „Indoor Mapping,“ [Elektronski]. Dosegljivo: http://wiki.openstreetmap.org/wiki/Indoor_Mapping. [Poskus dostopa 2015 januar].

[12] OpenStreetMap, „Simple Indoor Tagging,“ [Elektronski]. Dosegljivo: http://wiki.openstreetmap.org/wiki/Simple_Indoor_Tagging.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest in C. Stein, „Introduction to algorithms,“ London, The MIT Press, 2009, str. 658-664.

[14] B. Paramvir in V. N. Padmanabhan, „RADAR: An in-building RF-based user location and tracking system,“ v *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, Tel Aviv, 2000.

[15] H. Wang, S. Sen, A. Elgohary, M. Farid, M. Youssef in R. R. Choudhury, „No need to war-drive: unsupervised indoor localization.,“ v *Proceedings of the 10th international conference on Mobile systems, applications, and services*, Low Wood Bay, 2012.

[16] M. Yan, G. Nakibly, A. Schulman in D. Boneh, „PowerSpy: Location Tracking using Mobile Device Power Analysis,“ *arXiv preprint,* 2015.

[17] Joyent, Inc, „Node.js,“ [Elektronski]. Dosegljivo: http://nodejs.org/. [Poskus dostopa februar 2015].

Kazalo slik